Язык Си в примерах

Материал из Викиучебника — открытых книг для открытого мира

Этот учебник по C (то есть — по языку программирования Си) совместит:

- примеры обращения с основными программными структурами и процедурами;
- образцы решения распространённых алгоритмических задач на Си.

Выбор образцов стремится к таким, для каких устоялся небольшой и бесспорный инструментарий; так, что на одной странице можно привести все стоящие внимания решения: простое — но не совсем правильное, правильное — но слегка непривычное, эффективное — но длинное…

Гипертекстовая форма этого *вики*учебника позволит авторам ненавязчиво отсылать читателя к предварительным задачам и примерам, желательным для целостного понимания более сложного примера; об уровне сложности мы также стремимся уведомлять.

Для преподавателей программирования силён соблазн измышлять непрактичные, «игрушечные» задачи, укрощающие сложность и запутанность требуемого решением кода. Не предавая их анафеме, мы всё же предпочтём таким задачам давать более жизненные примеры, порою знакомя читателя с сопутствующими языку Си технологиями (GNU Bison, Make, и т. д…)

О правильном стиле кодирования на Си (оформление, именование переменных и функций, стили вызовов функций, когда и как использовать макросы…) мнения разнятся. В этом учебнике авторами за основу выбран стиль, принятый в учебнике «Язык программирования Си» Брайана Кернигана и Дениса Ритчи.

См. также

- Реализации алгоритмов
- http://users.powernet.co.uk/eton/kandr2/ — решения примеров из учебника «Язык программирования Си».
- Вводный учебник по объектно-ориентированному программированию на языке Си++.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах&oldid=145924

Эта страница в последний раз была отредактирована 24 апреля 2018 в 14:29.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Компиляция программ

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Программа на языке Си — один или несколько текстовых файлов, которые также называются исходными.

Исполнить исходные файлы нельзя, их необходимо *скомпилировать*, т.е. создать исполняемый файл, содержащий в себе инструкции процессора и пригодный для запуска на компьютере.

Процесс преобразования исходных файлов в исполняемый называется *компиляцией*. Если ваша программа состоит из одного исходного файла `hello.c`, то для его компиляции компилятором GNU C достаточно выполнить команду:

```
bash$ gcc hello.c -o hello
```

Если не получается или будете создавать папку для проекта, то разместите её поближе к корневой пользователя /home/username/folder/hello.c и запустите команду:

```
[username@localhost ~]$ gcc ~/development/hello.c -o hello
```

В результате получится файл `hello`, имя которого мы указали в опции `-o`. Этот файл является исполняемым и его можно запускать (**exe**cute) при помощи команды:

```
bash$ ./hello
```

Пара символов `./` перед `hello` означает "искать исполняемый файл `hello` в текущей директории".

Строчка

```
bash$ gcc xxx.c yyy.c -o zzz -I./common -I.. -lm
```

соответствует команде: "скомпилировать файлы xxx.c yyy.c в программу zzz; заголовочные файлы находятся в директориях ./common и ..; подключить библиотеку libm"

Библиотека libm (подключаемая с помощью опции `-lm`) содержит откомпилированные математические функции, которые объявляются в заголовочном файле `math.h`. Если вы используете функции из этой библиотеки (такие как `log`, `sin`, `cos`, `exp`), то не забывайте подключать её при компиляции.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

Подробную информацию об опциях компилятора gcc можно получить, если набрать

```
bash$ man gcc
```

или

```
bash$ info gcc
```

Ссылки

- http://digteh.ru/MCS51/C51/c51.php - Язык программирования СИ для микроконтроллеров
- http://www.christian-heffner.de - Бесплатная среда разработки для windows со встроенным компилятором

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Компиляция_программ&oldid=92007

Эта страница в последний раз была отредактирована 17 мая 2014 в 14:22.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Простейшая программа «Hello World»

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Содержание

Вариант «простой»

Вариант «классический»

Вариант «экзотический»

См. также

Примечания

Вариант «простой»

Первая программа, которую мы рассмотрим, — это «Hello World» — программа, которая выведет на экран строку текста «Hello, World!» («Здравствуй, мир!») и закончит своё выполнение.

```
#include <stdio.h>

int main (void)
{
  puts ("Hello, World!");
  return 0;
}
```

Посмотрим на неё внимательно. Первая строка — `#include <stdio.h>` — означает «включи *заголовок* `stdio.h`». В этом заголовке *объявляются* функции, связанные с вводом и выводом данных.[1][2]

Аббревиатуру *stdio* можно перевести как *стандартный ввод-вывод* (англ. standard input/output). Буква «h» после точки означает *заголовок* (англ. header). В заголовках (которые *как правило* представлены отдельными *заголовочными файлами*) обычно объявляются предоставляемые соответствующими им библиотеками функции, типы данных, константы и определения препроцессора.[3]

Далее идёт *определение* функции `main`. Оно начинается с *объявления*:

```
int main (void)
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в

что значит: «функция с именем `main`, которая возвращает целое число (число типа `int` от англ. integer) и у которой нет аргументов `(void)`»

формате ADX

В качестве варианта, стандарт допускает определение функции `main` как функции двух аргументов (`int argc` и `char *argv[]` — имена, разумеется, могут быть произвольными), что используется для получения доступа к *аргументам командной строки* из программы. В данном случае, эта возможность не требуется, поэтому функция определена как безаргументная (что также явно разрешено стандартом.)[4]

Английское слово void можно перевести как «пустота». Далее открываются фигурные скобки и идёт *тело* функции, в конце фигурные скобки закрываются. Функция `main` — главная функция программы, именно с нее начинается выполнение программы.

Тело функции, в свою очередь, определяет последовательность действий, выполняемых данной функцией — логику функции. Наша функция выполняет одно единственное действие:

```
  puts ("Hello, World!");
```

Это действие, в свою очередь, есть вызов функции `puts` стандартной библиотеки.[5] В результате выполнения этого вызова, на стандартный вывод (которым, скорее всего, окажется экран или окно на экране) печатается строка `Hello, world!`.

Затем идёт команда `return 0;`, которая завершает выполнение функции с возвратом значения 0, определяемого стандартом (для функции `main`) как *код успешного завершения*.[6][7]

Вариант «классический»

Этот вариант отличается использованием функций `printf` (вместо `puts`) и `getchar`.

```
#include <stdio.h>

int main (void)
{
  printf ("Hello, World!\n");
  getchar ();
  return 0;
}
```

В отличие от функции `puts`, выводящей переданную в качестве аргумента символьную строку, первый и обязательный аргумент функции `printf` определяет *формат* вывода.[8]

В общем случае, формат состоит из произвольного текста (не включающего символ %) «перемешанного» с *указателями преобразований* (предваряемыми символом %). В данном случае, однако, эта возможность не используется и никаких преобразований не выполняется.

Обратите внимание на появившуюся в *строковой константе* комбинацию `\n` — она включает в выводимую строку управляющий код (или *управляющий символ*) перевода (также *разрыва* или *завершения*) строки. В отличие от функции `puts`, всегда добавляющей этот код к выводимой строке, `printf` требует явного его указания.

Действующая редакция стандарта определяет *семь* таких комбинаций, причем все они записываются с помощью символа обратной косой черты \ (см. ASCII коды символов).[9]

Обратим внимание и на следующее новшество:

```
  getchar ();
```

Окружение, в котором запускается программа, как правило можно настроить так, что вывод программы будет оставаться на экране после ее выполнения неограниченно долго. Проще всего это обеспечить вызывая программу из командного интерпретатора (который, в свою очередь, может быть запущен в окне эмулятора терминала) или (в зависимости от системы) окна Cmd.exe.

Однако, при запуске непосредственно из графического окружения, отведенное программе окно может закрыться сразу же после завершения программы. Функция `getchar`[10] ожидает ввод пользователя, тем самым «откладывая» завершение программы (`return`). Какие именно действия могут прервать это ожидание — зависит от системы, однако можно надеяться, что нажатие клавиши ⏎ Enter завершит эту функцию в любой системе.

В некоторых руководствах для этой же цели предлагается функция `getch`. Однако, эта функция (в отличие от `getchar`) не является *стандартной* и, к тому же, зависима от платформы. Так, в некоторых системах использование `getch` требует включения файла `curses.h` и предшествующего вызова функции `initscr`. В других системах, однако, `getch` может быть объявлена в `conio.h`, и выполнение `initscr` — не требуется.

Вариант «экзотический»

Наконец, рассмотрим следующий, выходящий за рамки стандарта, вариант этой программы.

```
#include <windows.h>

int
main (void)
{
  MessageBox (0, "Hello, World!",
              "Message", MB_OK);
  return 0;
}
```

Использование заголовка `windows.h` может произвести впечатление приемлемости этого варианта кода в рамках только лишь одной конкретной системы. Однако, для использованного здесь интерфейса существует и независимая свободная реализация — Wine, — позволяющая собрать и выполнить данный вариант на таких системах, как, например, GNU/Linux, FreeBSD, Solaris и Mac OS X.

См. также

- C11
- Wine

Примечания

1. В действительности, `#include <...>` — это директива препроцессора, то есть команда, которая выполняется до начала компиляции файла. Смысл этой директивы очень прост и заключается в том, чтобы на место, где указана эта директива, вставить содержимое файла, имя которого указано в угловых скобках. Обычно заголовочные файлы содержат только прототипы функций, то есть просто список функций с указанием аргументов и типа возвращаемого значения.
2. *6.10.2 Source file inclusion (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
3. *7.21 Input/output `stdio.h` (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
4. *5.1.2.2.1 Program startup (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
5. *7.21.7.9 The puts function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
6. *5.1.2.2.3 Program termination (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
7. *7.22.4.4 The exit function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
8. *7.21.6.1 The fprintf function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
9. *5.2.2 Character display semantics (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
10. *7.21.7.6 The getchar function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Простейшая_программа_«Hello_World»&oldid=148335

Эта страница в последний раз была отредактирована 13 сентября 2018 в 08:22.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Учимся складывать

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Разнообразные вычисления — моделирование, решение алгебраических и дифференциальных уравнений — это то, для чего и создавались первые компьютеры. Давайте и мы научимся использовать компьютер для вычислений. Начнём со сложения двух чисел.

Дано
: два разделенных *пробельными символами* целых чисел (в «текстовом» десятичном представлении) на стандартном вводе программы.

Найти
: значение суммы (разности, произведения, частного от деления — если имеет смысл) этих двух чисел.

Новые элементы языка
: переменные; тип double; операторы &, *, ? : контекста *выражения* и оператор if контекста *утверждения*; макроподстановка assert, функции printf, scanf.

Содержание

Вариант «простой»

Вариант «дробный»

Л-значения и ссылки

Вариант «произвольный»

Вариант «арифметический»

Вариант «тернарный»

Примечания

Вариант «простой»

В отличие от рассмотренной ранее простейшей программы, в данной задаче нам потребуются переменные — ячейки памяти, в которые функция ввода сохранит введенные пользователем числа, подлежащие сложению.

Как и в случае функций (в том числе `main`), объявление переменных в Си предполагает указание их типов. Основные числовые типы языка — `int` (целое число фиксированной разрядности) и `double` (число с плавающей запятой.) Поскольку мы уже использовали тип `int` в описании функции `main`, применим его же в данной задаче.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

```c
#include <assert.h>
#include <stdio.h>

int main ()
{
  int a, b;
  int r = scanf ("%d%d", &a, &b);
  assert (r == 2);
  printf ("%d\n", a + b);
  return 0;
}
```

Рассмотрим выполнение этой программы почти с ее завершения — вызова функции `printf`.[1] Данная функция выведет целое число в десятичной форме (согласно *указателю преобразования* `%d`), завершая вывод переводом строки (`\n`).

Число, которое будет выведено, является результатом вычисления выражения `a + b` — или же, проще говоря, — суммой значений переменных *a* и *b*.

Вводимые пользователем числа помещаются в эти переменные функцией `scanf`, вызываемой в коде выше как `scanf ("%d%d", &a, &b)`.[2] Здесь, как и в `printf`, используется указатель преобразования `%d`, означающий на этот раз *считывание* числа в десятичной форме (возможно — предваряемого пробелами). Поскольку указатель повторен дважды, два числа будут считаны и помещены в упомянутые в аргументах две переменные *a* и *b*. (Необходимый здесь унарный оператор `&` оставим пока без внимания.)

Как и `printf`, функция `scanf` *объявлена* в *заголовке* (англ. header) `stdio.h`.[3]

В простейшей программе от действий пользователя не зависело ровным счетом ничего. Каким, однако, будет результат выполнения данной программы, если ввод пользователя *не будет* начат двумя числами в десятичной форме?

Для проверки соответствия ввода пользователя требованиям программы мы *сохраняем* (`=`) результат выполнения `scanf` — количество успешно измененных переменных — в целочисленной переменной с именем *r* (`int r`), после чего *требуем* равенства ее значения двум (`assert (r == 2);`.)

Действие *макроподстановки* `assert` заключается в *вычислении* выражения, переданного ей первым (и единственным) аргументом и *аварийном завершении* программы в случае, если полученное значение — ноль («логическая ложь».)[4]

Наконец, в самом начале функции `main` *определены* (помимо упомянутой уже *r*) целочисленные переменные *a* и *b*. Их значение в начале выполнения функции `main` может быть произвольным,[5] но после успешного (что проверяется использованием `assert`) завершения `scanf` они будут содержать два числа, которые удалось последовательно считать со стандартного ввода.

Вариант «дробный»

Рассмотренную выше программу несложно изменить для использования чисел с плавающей запятой.

```c
#include <assert.h>
#include <stdio.h>

int
main ()
{
  double a, b;
  int r = scanf ("%lg%lg", &a, &b);
  assert (r == 2);
  printf ("%lg\n", a + b);
  return 0;
}
```

Как видно, в этом случае изменяются лишь тип переменных *a*, *b* (`int` → `double`) и указатели преобразований (%d → `%lg`.)

Здесь следует отметить, что в случае `scanf` совершенно идентично будут действовать указатели преобразований `%lg`, `%lf` и `%le`. Напротив, в случае `printf` не будет разницы между `%lg` и `%g`. Причины такого поведения мы также пока оставим без внимания.

Желающим изучить использование других числовых типов в этой задаче предлагается обратиться к разделу «Числовые типы» приложения.

Л-значения и ссылки

Вызов функции в чем-то подобен отправке письма. Мы указываем на конверте имя функции, вкладываем в конверт копии значений, которые хотим передать, и отправляем письмо (вызываем функцию).

В некоторых случаях, однако, мы хотим получить от функции что-либо обратно. Иногда, для этого достаточно предусмотренного языком *возвращаемого значения*. Иначе, можно отправить функции своего рода *конверт с обратным адресом* (один или несколько). Именно для этого — для получения *адреса л-значения* — служит унарный оператор &.

Так, запись `&a` означает *адрес ячейки памяти, выделенной для хранения значения переменной* `a`, или же, проще, — *ссылка* (англ. reference) на переменную `a`.

Справа от оператора & указывается *л-значение* (англ. lvalue.[6] Не вдаваясь в подробности отметим, что под л-значением понимают любое выражение, которое может стоять *слева* от оператора присваивания `=`. В частности, л-значением является любое выражение, состоящие из имени действительной в данном контексте (англ. scope) переменной.

Для обращения к ячейке памяти по ссылке используется унарный оператор `*`. Так, выражение `*(&a) = 1` полностью равнозначно `a = 1`.

Вариант «произвольный»

Стандарт C11 требует от реализации языка поддержки числовых типов разрядности 8, 16, 32 и 64 бит.[7] Если по каким-либо причинам в вычислениях требуется бóльшая разрядность, следует использовать библиотеки операций с числами *произвольной* разрядности — как, например, GNU MP.[8]

В частности, благодаря GNU MP, пример ниже выведет сумму двух введенных чисел даже если любое из них — или результат сложения — превышает 2^{64}.

Код имеет следующие отличия от исходного варианта:

1. ввод-вывод чисел произвольной разрядности выполняется функциями `gmp_scanf` и `gmp_printf` — *определенными* GNU MP и *объявленными* в `gmp.h`; заголовок `stdio.h` не требуется;
2. вместо оператора сложения `+` используется вызов функции `mpz_add (a, a, b)`; в такой форме, действие функции подобно действию выражения `a += b` для переменных «встроенных» целочисленных типов;
3. переменные GNU MP требуют обязательной инициализации, которая в примере выполнена функцией `mpz_inits ()`;
4. тип используемых для вычисления переменных — `mpz_t`.

```c
#include <assert.h>
#include <gmp.h>

int
main (void)
{
  mpz_t a, b;
  mpz_inits (a, b, 0);
  int r
    = gmp_scanf ("%Zd%Zd", a, b);
  assert (r == 2);
  mpz_add (a, a, b);
  gmp_printf ("%Zd\n", a);

  return 0;
}
```

Вариант «арифметический»

В программе ниже мы кроме суммы вычислим разность и произведение введенных чисел, а также, если второе число отлично от нуля, — частное от деления первого числа на второе.

```c
#include <assert.h>
#include <stdio.h>

int
main ()
{
  double a, b;
  int r
    = scanf ("%lg%lg", &a, &b);
  assert (r == 2);
  if (b != 0) {
    printf (("a + b = %lg\n"
             "a - b = %lg\n"
             "a * b = %lg\n"
             "a / b = %lg\n"),
            a + b, a - b, a * b, a / b);
  } else {
    printf (("a + b = %lg\n"
             "a - b = %lg\n"
             "a * b = %lg\n"),
            a + b, a - b, a * b);
  }
  return 0;
}
```

В этой программе нам потребовался *условный оператор контекста утверждения* (англ. if statement) — один из четырех (наряду с &&, `||` и `? :`) условных операторов языка. Его синтаксис:[9]

```
if (выражение)
  тело-если-истинно

if (выражение)
  тело-если-истинно
else
  тело-если-ложно
```

Где *тело-если-истинно* и *тело-если-ложно* могут быть (каждый) единственным утверждением (завершаемым `;`), или же, как в примере выше, — `{ }`-блоком.

В случае, если результат вычисления *выражения* — истина (другими словами — отличен от 0), выполняется *тело-если-истинно*; в противном случае (и если используется `else`) — *тело-если-ложно*.

Заметьте, что каждое простое *утверждение* (англ. statement) завершается точкой с запятой. Одна из самых популярных синтаксических ошибок начинающих программистов — не ставить точку с запятой после утверждений.

Вариант «тернарный»

Используя условный оператор контекста *выражения* (англ. conditional operator), а также приняв во внимание тот факт, что `printf` проигнорирует «избыточные» аргументы (аргументы сверх количества, требуемого указателями преобразований в указанной первым аргументом строке формата), можно незначительно сократить код предыдущего примера.

```
#include <assert.h>
#include <stdio.h>

int
main ()
{
  double a, b;
  int r
    = scanf ("%lg%lg", &a, &b);
  assert (r == 2);
  printf ((b != 0
           ? ("a + b = %lg\n"
              "a - b = %lg\n"
              "a * b = %lg\n"
              "a / b = %lg\n")
       : ("a + b = %lg\n"
              "a - b = %lg\n"
              "a * b = %lg\n")),
          a + b, a - b, a * b,
      (b != 0 ? a / b : 42));
  return 0;
}
```

В этом варианте, код вновь содержит те же самые строковые константы, что и в предыдущем, однако используется лишь один вызов функции `printf` — первым аргументом которой (в зависимости от значения переменной `b`) окажется одна из этих констант.

Число 42, которое передается `printf` пятым аргументом в случае нулевого значения *b*, может быть произвольным. Поскольку в этом случае функция `printf` получает строку формата лишь с *тремя* указателями формата, фактически использованы будут лишь аргументы с первого (формат) по четвертый (произведение *a* и *b*).

Общий синтаксис тернарного оператора `? :` следующий:[10]

```
выражение ? если-истинно : если-ложно
```

Где *выражение* вычисляется первым, после чего, если оно истинно (отлично от 0), вычисляется — и принимается результатом оператора в целом — выражение *если-истинно*; в противном случае — *если-ложно*.

Примечания

1. *7.21.6.1 The fprintf function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
2. *7.21.6.2 The fscanf function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
3. *7.21 Input/output `stdio.h` (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
4. *7.2.1.1 The assert macro (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
5. *6.7.9 Initialization (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
6. *6.3.2.1 Lvalues, arrays, and function designators (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
7. *7.20.1.2 Minimum-width integer types (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
8. *The GNU MP Bignum Library (англ.) Проверено 2015-04-04 г.*
9. *6.8.4.1 The if statement (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
10. *6.5.15 Conditional operator (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Учимся_складывать&oldid=145243

Эта страница в последний раз была отредактирована 14 марта 2018 в 05:50.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Максимум

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Дано
: непустая последовательность разделенных *пробельными символами* целых чисел (в «текстовом» десятичном представлении) на стандартном вводе программы. Последовательность конечна, но ее длина заранее неизвестна.

Найти
: предельные значения последовательности (минимум, максимум) и основные статистики (среднее, среднеквадратичное отклонение.)

Указания
: воспользуйтесь циклом «пока» (`while`) для чтения последовательности и функциями `pow`, `sqrt` для вычисления среднеквадратичного отклонения.

Содержание

Решение

Вычисление и вывод результатов

Чтение и накопление

Условия корректности ввода

Цикл «пока»

Задания

См. также

Примечания

Решение

```
#include <assert.h>
#include <math.h>
#include <stdio.h>

int
main ()
{
  int n = 1, cur, min, max;
  double sum, sum_sq;

  int r
    = scanf ("%d", &cur);
  assert (r == 1);
  sum = min = max = cur;
  sum_sq = pow (cur, 2);

  while (1 == (r = scanf ("%d", &cur))) {
    if (cur < min) {
      min = cur;
    } else if (cur > max) {
      max = cur;
    }
```


Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

```
    ++n;
    sum     += cur;
    sum_sq  += pow (cur, 2);
  }
  assert (r == EOF);
  assert (! ferror (stdin));

  printf (("Range:  [%d, %d]\n"
           "Count:  %d\n"
           "Mean:   %lg\n"
           "StdDev: %lg\n"),
          min, max, n, sum / n,
          sqrt (sum_sq / n - pow (sum / n, 2)));

  return 0;
}
```

Вычисление и вывод результатов

Разбор программы вновь начнем с ее завершения — вызова функции `printf`[1] для вывода результатов ее работы. В данном случае, мы выводим значения переменных *min*, *max*, *n* (обновляемых после успешного чтения каждого очередного элемента числовой последовательности.) Среднее значение считаем как отношение *суммы* всех элементов (*sum*) к их *количеству* (*n*).

Среднеквадратичное отклонение вычислено как корень из дисперсии, которая, в свою очередь, вычислена по правилу «среднее квадратов минус квадрат среднего» — `sum_sq / n - pow (sum / n, 2)` — тривиально следующему из ее определения. Значение используемой нами здесь функции `pow` — первый аргумент, возведенный в степень, заданную вторым аргументом; значение `sqrt` — квадратный корень единственного аргумента. Эти функции является частью *стандартной библиотеки* и *объявлены* в *заголовке* `math.h`.[2][3]

Чтение и накопление

Чтение последовательности выполнено в два этапа. Вначале, считанный первым элемент используется для инициализации *предельных значений* (*min*, *max*) и *сумм* (*sum*, *sum_sq*).

Затем, каждый очередной элемент (все так же считываемый функцией `scanf`[4]) подвергается следующей обработке:

1. сравнивается с текущими значениями переменных *min*, *max*; если элемент лежит вне этих пределов — один из них изменяется соответственно;
2. значение *суммы* (*sum*) увеличивается на величину текущего элемента; значение *суммы квадратов* (*sum_sq*) — на величину квадрата;
3. кроме того, успешное чтение элемента приводит к *инкременту* (увеличению на 1) хранящей количество считанных элементов переменной *n*.

Каждый вызов `scanf ("%d", &cur)` считывает целое число в десятичной записи (%d) в переменную `cur`. *Предшествующие* числу *пробельные символы* (пробел, табуляция, перевод строки) при этом игнорируются, а значит могут быть использованы — в любых сочетаниях — для *разделения* элементов последовательности.

Условия корректности ввода

После завершения чтения последовательности, мы *требуем* (используя уже известную нам макроподстановку `assert`[5]) истинности следующих двух условий, смысл которых сводится к тому, что чтение последовательности *не было прервано* ни появлением во входном потоке каких-либо «нечисловых» данных, ни ошибкой ввода-вывода.

1. Мы проверяем равенство последнего возвращенного функцией `scanf` значения константе (макроопределению) `EOF` (*конец файла*; англ. end of file), что соответствует или *исчерпанию* нашего входного потока (*стандартного ввода*), или возникновению ошибки ввода-вывода. Другое возможное на данном этапе значение — ноль — укажет на наличие во входном потоке данных, которые не были опознаны функцией `scanf` как требуемое условием целое число в десятичной записи (%d).
2. Если предыдущее требование выполняется (иными словами — останов произошел *не* из-за «нечислового» ввода), мы также явно требуем *ложности* значения функции *признака ошибки* `ferror` для стандартного ввода (`stdin`.)[6]

Цикл «пока»

При чтении элементов последовательности начиная со второго мы используем оператор цикла «пока» `while`, *синтаксис* которого напоминает синтаксис рассмотренного ранее *условного оператора контекста утверждения* `if`:[7]

```
while (выражение)
  тело
```

Подобно оператору `if`, если результат вычисления *выражения* — ложь (другими словами — 0), *тело* не выполняется и *реализация* языка Си просто переходит к следующему *утверждению* (англ. statement) кода.

Если же результат — истина (отличен от 0), выполняется *тело*, после чего реализация *вновь возвращается* к началу цикла.

В нашем случае, *условием продолжения* выполнения цикла является равенство единице количества успешно считанных `scanf` значений. Тем самым, цикл можно прочитать следующим образом:

> *пока* удается считать очередное значение?
>
> > обновляй переменные *min*, *max*, *n*, *sum*, *sum_sq*.

Задания

1. Измените программу для вычисления предельных значений и статистик последовательности чисел с плавающей запятой.
2. В условии задачи мы потребовали наличия в последовательности хотя бы одного элемента. Определите, какие сложности возникнут при попытке обобщить данную программу на случай *пустой* последовательности?
3. Дополните программу выводом «промежуточных» значений — суммы элементов последовательности и суммы их квадратов.

4. Реализуйте также вычисление и вывод моментов случайной величины вплоть до четвертого и, на их основе, — коэффициента асимметрии и коэффициента эксцесса.

См. также

- Математическое ожидание
- Среднеквадратическое отклонение

Примечания

1. *7.21.6.1 The fprintf function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
2. *7.12.7.4 The pow functions* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
3. *7.12.7.5 The sqrt functions* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
4. *7.21.6.2 The fscanf function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
5. *7.2.1.1 The assert macro* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
6. *7.21.10.3 The ferror function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
7. *6.8.5.1 The while statement* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Максимум&oldid=137123

Эта страница в последний раз была отредактирована 8 мая 2017 в 01:53.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Таблица умножения

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Дано
: положительное целое число n (в «текстовом» десятичном представлении), переданное программе как аргумент командной строки. Значение n^2 не превышает максимального допустимого значения для типа `long` (*LONG_MAX*.)

Надо
: вывести на стандартный вывод таблицу умножения размера $n \times n$.

Указания

- Начать разработку программы можно с цикла ($j = 1, \ldots, n$) вывода некоторой (i-ой) строки таблицы. Затем, этот цикл (*цикл значений*) можно в свою очередь заключить в *цикл строк* ($i = 1, \ldots, n$.)
- Для выравнивания столбцов таблицы имеет смысл воспользоваться формулой *длины десятичного представления* положительного целого числа ($x \geqslant 0$) — $w = 1 + \lfloor \lg(x) \rfloor = \lceil \lg(1 + x) \rceil$ (где $\lfloor \alpha \rfloor$ и $\lceil \alpha \rceil$ — операции округления «к нулю» и «от нуля», соответственно.)

Дополнительно
: включите в программу интерпретацию аргумента (n) согласно правилам языка Си для числовых констант. Реализуйте вывод таблицы умножения в шестнадцатеричном, восьмеричном или десятичном представлении в зависимости от способа записи аргумента командной строки. (Так, например, аргумент `0x12` должен приводить к выводу таблицы умножения 18 × 18 в шестнадцатеричном представлении.)

Новые элементы языка
: форма записи *символьных констант*; операторы continue, for; функции ceil, isdigit, log, putchar, strchr, strtol; *указатель разрядности* `*` форматного преобразования; двуаргументная форма объявления функции `main`.

Содержание

Решение

Цикл строк

Цикл значений

Вычисление предельной разрядности

Разбор аргументов командной строки

Видимость переменных

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

Вариант «мультисистемный»

Задания

Примечания

Решение

```c
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int
main (int argc, char *argv[])
{
  assert (argc - 1 == 1);
  long n;
  {
    char *tail;
    n = strtol (argv[1], &tail, 0);
    assert (tail > argv[1]);
    assert (*tail == '\0');
  }

  assert (n >= 1);
  assert (n <= sqrt (LONG_MAX));
  const int width = (int)ceil (2 * log (1 + n) / log (10));

  long i;
  for (i = 1; i <= n; i++) {
    long j;
    for (j = 1; j <= n; j++) {
      printf (" %*ld", width, i * j);
    }
    putchar ('\n');
  }

  return 0;
}
```

Примета работы программы (построение таблицы умножения для $n = 5$):

```
$ ./multable 5
  1  2  3  4  5
  2  4  6  8 10
  3  6  9 12 15
  4  8 12 16 20
  5 10 15 20 25
$
```

Ниже мы разберем работу данной программы, начиная с завершающих ее *циклов вычисления и вывода результатов* и двигаясь далее к началу.

Цикл строк

В данном примере мы используем оператор цикла `for`, являющийся обобщением уже рассмотренного нами в разделе Максимум оператора `while`.[1] А именно, запись `while (`*`выражение`*`)` *`тело`* равносильна `for (; `*`выражение`*`; )` *`тело`*. С другой стороны, *цикл строк* в примере выше можно было бы записать с использованием `while` следующим образом:

```c
  long i = 1;
  while (i <= n) {
```

```
    код-вывода-строки;
    i++;
  }
```

Иными словами, для первой итерации цикла мы *присваиваем* переменной *i* значение 1. Затем, мы выводим очередную (*i*-ую) строку *пока i* не превысит заданного пользователем значения *n*. Завершает *тело* цикла *инкремент* (увеличение на 1) номера строки *i*.

С точки зрения языка, содержимое «скобок» оператора `for` состоит из трех разделенных точкой с запятой (`;`) частей, каждая из которых может быть или *пустой*, или *совершенно произвольным* выражением; например: `i = 1`, `f (42)`, `x *= 3`, `a = b = c`. При этом:

1. первое выражение — *инициализация* — вычисляется *перед началом цикла*, — как если бы было записано непосредственно перед самим оператором `for`;
2. второе выражение — *условие* — вычисляется *перед каждой итерацией*, — совершенно аналогично условию для оператора `while`;
3. наконец, третье выражение — *изменение* — выполняется *в конце* («после») каждой итерации.

Можно было бы сказать, что оператор `for` является *избыточным* при наличии `while`, однако в языке существует также оператор `continue`, *досрочно завершающий* текущую итерацию[2] (с переходом к проверке условия и, при его выполнении, — к следующей итерации.) При использовании `while`, выражение «изменения» («перехода к следующей итерации») пришлось бы дублировать возможно перед каждым случаем использования оператора `continue`. Напротив, в случае `for`, выражение *изменения* выполняется после *любой* итерации — даже если таковая завершена досрочно.

Цикл значений

Телом *цикла строк* должна стать, очевидно, некая процедура вывода заданной строки, которую мы вновь оформляем как цикл — использующий уже новую переменную *j*:

```
    long j;
    for (j = 1; j <= n; j++) {
      printf (" %*ld", width, i * j);
    }
    putchar ('\n');
```

В данном коде, мы встречаемся с двумя новыми возможностями языка. Первая из них — указатель `*` *переменной разрядности* преобразования в *строке формата* функции `printf`.[3] В данном случае, наличие этого указателя (в форме `%*ld`) означает, что вместо одного аргумента, функция `printf` использует для форматного преобразования *два* (а именно: *width* и произведение $i \times j$.) При этом, первый аргумент задает *разрядность* преобразования, второй — само подлежащее преобразованию (и выводу) значение.

Вторая возможность — функция `putchar`, выводящая на стандартный вывод заданный символ (код);[4] в данном случае, `'\n'` — управляющий код (или *управляющий символ*) перевода (*разрыва*, *завершения*) строки. **Обратите внимание** на использование *одинарных кавычек* при записи *символьных констант* (в отличие от *строковых* — для которых используются *двойные кавычки*.)

Если бы было заранее известно, что значение произведения не превышает, например, 9999, можно было бы обойтись фиксированной разрядностью (`" %4ld", i * j` — или, что то же, — `" %*ld", 4, i * j`.) Однако коль скоро в задаче требуется обрабатывать широкий диапазон чисел (наибольшее значение типа `long` согласно стандарту C11 должно быть не менее порядка 2×10^9), разрядность результата необходимо вычислять исходя из заданного пользователем значения *n*.

Поскольку мы также предваряем дополнительным *пробелом* каждое выводимое число, общая ширина каждого столбца окажется на единицу большей, чем используемое значение разрядности *width*. Вычислению последнего, в свою очередь, посвящен следующий раздел.

Вычисление предельной разрядности

Для получения ровных столбцов в выводе программы, каждое выводимое значение должно состоять из равного количества знаков. Ясно, что из всех возможных произведений $i \times j$, $i = 1, \ldots, n$, $j = 1, \ldots, n$, максимальное значение имеет $n \times n = n^2$, а значит именно количество знаков в записи этого числа и будет предельным для таблицы умножения $n \times n$.

Далее, мы пользуемся тем фактом, что *позиционная* (в частности — десятичная) запись числа является его *разложением* по степеням *основания системы счисления*. Ясно, что максимальная степень *n* в разложении положительного целого числа *x* по степеням основания *b* равна $\lfloor \log_b x \rfloor$, а общее число разрядов (включая *единицы* — при нулевой степени основания) — на единицу больше *n*.

Для получения конечной формулы, нам остается лишь воспользоваться следующими простыми свойствами логарифмов:

- $\log_b (x^n) = n \log_b x$;
- $\log_b x = \log_a x / \log_a b = \ln x / \ln b$.

В итоговом выражении — `(int)ceil (2 * log (1 + n) / log (10))` — мы используем функции `ceil` и `log`, значением которых являются результат округления «от нуля» (в сторону роста *абсолютного значения*) и натуральный логарифм аргумента, соответственно. Эти функции являются частью *стандартной библиотеки* и *объявлены* в *заголовке* `math.h`.[5][6]

Ясно, что данное выражение теряет смысл при $n < 0$. Кроме того, при $n = 0$, реализованные нами циклы вычисления—вывода сформируют *пустой* результат, который также *можно* считать ошибочным. Чтобы этого избежать, перед вычислением искомого выражения мы *требуем* (макроподстановкой `assert`[7]) выполнения условия `n >= 1`.

Здесь же мы требуем, чтобы значение *n* не превышало квадратного корня из максимального допустимого значения для используемого нами типа `long` (макроподстановка *LONG_MAX* — *объявлена* в *заголовке* `limits.h` *стандартной библиотеки*.) В противном случае, один или более из результатов умножения может *переполнить* разрядность данного типа. (Отметим, впрочем, что формирование таблицы умножения теряет практический смысл уже при много меньших значениях *n*.)

Разбор аргументов командной строки

Наконец, рассмотрим получение в программе переданного как аргумент командной строки значения *n*.

```
  assert (argc - 1 == 1);
  long n;
  {
    char *tail;
    n = strtol (argv[1], &tail, 0);
    assert (tail > argv[1]);
    assert (*tail == '\0');
  }
```

Данный фрагмент кода начинается *требованием* (макроподстановкой `assert`[7]) наличия *единственного* аргумента командной строки. Здесь следует отметить, что переменная *argc* отражает общую длину массива *argv*, в который входит — нулевым элементом — имя самой программы: `argv[0]`. Тем самым, условие наличия и единственности аргумента может быть записано как `argc == 2` — или же как приведено в коде.

В свою очередь, переменные *argc* и *argv* являются аргументами главной функции `main`, при *объявлении* ее следующим образом:[8]

```
int
main (int argc, char *argv[])
```

Мы объявляем переменную *n* и, в отдельном вложенном блоке, — указатель *tail*. Вызываем функцию `strtol` (*объявленную* в *заголовке* `stdlib.h`) преобразования строкового представления числа в «машинное» — значение типа `long`;[9] результат сохраняем в переменной `n`. Первый аргумент функции — строка, подлежащая преобразованию; третий — основание системы счисления, или 0 — для использования правил языка Си для числовых констант (например: `0x12` — 12_{16}; `012` — 12_{8}; `12` — 12_{10}.)

Функция `strtol` пропускает любые *пробельные символы* в начале строки (если они есть) и пытается разобрать следующее за ними число. При этом, указатель на первый же символ (код), не являющийся частью такого числа, помещается по адресу, переданному вторым аргументом (если не 0); в нашем случае — в переменную `tail`. Кроме того, если за *ведущими* пробельными символами *не следует* ничего похожего на число (в соответствии с указанной третьим аргументом формой записи), то по данному адресу помещается значение первого аргумента.

После выполнения данного преобразования, мы *требуем* истинности следующих двух условий, смысл которых сводится к тому, что первый аргумент командной строки содержит одно число (возможно — предваряемое ведущими *пробельными символами*) *и ничего после него*:

1. `tail > argv[1]` — некое число *было обнаружено* в данной строке (и извлечено из нее);
2. `*tail == '\0'` — за этим числом сразу же следует *признак конца строки* (код 0.)

В заключение отметим, что во-первых, можно, конечно, обобщить данную программу на случай произвольного количества аргументов — последовательно обрабатывая их в цикле. (В отсутствие аргументов можно выводить таблицу некоего явно определенного в коде размера; например: 10 × 10.) С другой стороны, возможны и другие обобщения.

Во-вторых, аргументы командной строки используются как правило для передачи данных *в дополнение* к передаваемым через файловые потоки, и в частности — стандартный ввод. Данную программу можно без значимого ущерба для функциональности или удобства переписать для получения значений *n* со стандартного ввода — применив цикл, аналогичный рассмотренному в разделе Максимум.

Видимость переменных

В примере выше мы «по мере необходимости», в различных точках программы, определили ряд требуемых нам переменных (*n*, *width*, *i*, *j*) — пользуясь тем, что стандарт (начиная с версии C11) позволяет определять переменные в любой точке кода, где допустимо *утверждение*. (В версии C99 и более ранних, определения переменных были допустимы только в начале *блока* — до первого утверждения.) Возникает вопрос: где именно в коде можно использовать каждую из определенных таким образом переменных?

Областью видимости (англ. scope) переменной (или *идентификатора* в общем) как правило оказывается часть содержащего переменную блока — от точки определения и до закрытия этого блока. Однако, вложенные блоки могут *скрывать* переменные вышестоящих блоков, объявляя свои собственные переменные с теми же именами. Так, например, функция `foo` в следующем примере выведет на стандартный вывод код перевода строки (из переменной *a* внутреннего блока) и *вернет* значение 1 (из одноименной переменной внешнего):

```
static int
foo (void)
{
  int a = 1;
  {
    int a = '\n';
    putchar (a);
  }

  return a;
}
```

Хорошей практикой представляется определение переменных так, чтобы область их видимости была *не большей*, чем строго необходимо. (Но не в ущерб ясности кода.) В этом случае, при перемещении (копировании) кода, случайное использование переменной там, где она не имеет смысла, с большей вероятностью будет диагностировано непосредственно реализацией языка.

Вариант «мультисистемный»

Используея нами для преобразования строкового представления числа в «машинное» функция strtol определяет *систему счисления* по *префиксу* строкового представления числа. В следующем примере мы частично воссоздаем эту логику для вывода таблицы умножения в шестнадцатеричном, восьмеричном или десятичном представлении в зависимости от способа записи аргумента командной строки.

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int
main (int argc, char *argv[])
{
  assert (argc - 1 == 1);
  long n;
  int base;
  const char *fmt;
  {
    char *tail;
    n = strtol (argv[1], &tail, 0);
    assert (tail > argv[1]);
    assert (*tail == '\0');

    char *p = strchr (argv[1], '0');
    if (p == 0
    || (p > argv[1] && isdigit (p[-1]))) {
      base = 10;
      fmt = " %*ld";
    } else if (p[1] == 'x' || p[1] == 'X') {
      base = 16;
      fmt = (p[1] == 'X' ? " %*lX" : " %*lx");
    } else {
      base = 8;
      fmt = " %*lo";
    }
  }

  assert (n >= 1);
  assert (n <= sqrt (LONG_MAX));
  const int width = (int)ceil (2 * log (1 + n) / log (base));

  long i;
  for (i = 1; i <= n; i++) {
    long j;
    for (j = 1; j <= n; j++) {
      printf (fmt, width, i * j);
    }
    putchar ('\n');
  }

  return 0;
}
```

Пример работы программы (построение таблицы умножения для $n = 5$ в шестнадцатиричном представлении):

```
$ ./multable 0X5
  1  2  3  4  5
  2  4  6  8  A
  3  6  9  C  F
  4  8  C 10 14
  5  A  F 14 19
$
```

По сравнению с исходным вариантом, пример дополнился следующим фрагментом кода:

```
    char *p = strchr (argv[1], '0');
    if (p == 0
    || (p > argv[1] && isdigit (p[-1]))) {
      base = 10;
      fmt = " %*ld";
    } else if (p[1] == 'x' || p[1] == 'X') {
      base = 16;
      fmt = (p[1] == 'X' ? " %*lX" : " %*lx");
    } else {
      base = 8;
      fmt = " %*lo";
    }
```

Кроме того, две из исходных *констант* (`10` при вычислении предельной разрядности и `" %*ld"` в качестве *строки формата* функции `printf`) заменены в данном варианте *переменными base* и *fmt*, соответственно.

Во фрагменте выше, мы используем функцию `strchr` (*объявленную* в *заголовке* `string.h`) поиска первого вхождения символа (кода) в строку[10] — в данном случае, поиска символа `0` в аргументе командной строки.

Функция возвращает *нулевой указатель* (0 или `NULL`) если символ не найден.[10] Поскольку это означает, что аргумент командной строки не содержит символа `0` *вовсе*, очевидно, что он не может содержать его и как часть префикса системы счисления (`0x` или `0`), а значит следует использовать десятичную систему счисления при вычислении предельной разрядности и *указатель представления* `d` — «десятичное число»: `base = 10; fmt = " %*ld";`.

Точно так же обрабатывается случай, когда найденному символу `0` *предшествует* любая десятичная цифра — `p > argv[1] && isdigit (p[-1])`. Используемая в этом условии функция `isdigit` (*объявлена* в *заголовке* `ctype.h`) возвращает *истину* (отличное от 0 значение) в том и только том случае, когда ее аргумент — десятичная цифра; в противном случае — возвращается 0.[11] Разумеется, обращение к *предшествующему* символу имеет смысл только в том случае, когда найденный символ `0` не является *первым* в строке — `p > argv[1]`.

Следующим мы проверяем условие `p[1] == 'x' || p[1] == 'X'` — наличие после `0` признака шестнадцатиричного представления `x` или `X`. В этом случае, *base* присваивается значение 16, *fmt* — `" %*lx"` или `" %*lX"` (в зависимости от регистра символа `x` в аргументе.)

Наконец, если условия выше (наличие цифры перед `0` *или* `x`, `X` — после) — не выполнены, мы присваиваем *base* значение 8, *fmt* — `" %*lo"`.

Логика выше не покрывает *всех* возможных значений аргумента командной строки, однако она вполне корректна — поскольку мы предполагаем успешное завершение функции strtol (используя макроподстановки `assert` выше — при получении значения *n*), что *уже* накладывает нужные нам дополнительные ограничения на запись данного аргумента.

Задания

1. Измените программу для вывода варианта таблицы в одной из трех альтернативных ориентаций: с обратным (от большего к меньшему) порядком строк; с обратным порядком столбцов; с обратным порядком и строк, и столбцов.
2. Дополните программу поддержкой *неположительных* значений *n*. А именно, для $n < 0$ следует строить таблицу умножения (n, …, -1) × (n, …, -1); для $n = 0$ — таблицу из единственного значения $0 \times 0 = 0$. (Указание: воспользуйтесь функцией `fabs`.[12])
3. Дополните программу поддержкой необязательного второго аргумента командной строки, используемого как значение основания системы счисления (2–16 — не предполагая использования кодовой таблицы ASCII, или же 2–36 в противном случае; по-умолчанию — использовать десятичную систему счисления.) Указание: используйте известность предельной разрядности результатов для определения символьного массива переменной длины. Вывести результирующую строку на стандартный вывод можно используя функцию `printf` — с указателем формата `%s`, или же функцию `fputs`[13] и предопределенный поток `stdout`.
4. Измените программу так, чтобы можно было выводить произвольный прямоугольный «фрагмент» таблицы умножения: (a, …, b) × (c, …, d).

Примечания

1. *6.8.5.3 The for statement (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
2. *6.8.6.2 The continue statement (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
3. *7.21.6.1 The fprintf function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
4. *7.21.7.8 The putchar function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
5. *7.12.9.1 The ceil functions (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
6. *7.12.6.7 The log functions (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
7. *7.2.1.1 The assert macro (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
8. *5.1.2.2.1 Program startup (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
9. *7.22.1.4 The strtol, strtoll, strtoul, and strtoull functions (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
10. *7.24.5.2 The strchr function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
11. *7.4.1.5 The isdigit function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
12. *7.12.7.2 The fabs functions (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
13. *7.21.7.4 The fputs function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Таблица_умножения&oldid=135355

Эта страница в последний раз была отредактирована 20 января 2017 в 13:50.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/ASCII-коды символов

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Дано
: произвольная кодовая (*октетная*) последовательность (возможно — в коде ASCII) на стандартном вводе программы. Последовательность конечна, но ее длина заранее неизвестна.

Надо
: вывести на стандартный вывод *дамп* последовательности — разделенные пробельными символами числовые значения считанных кодов.

Содержание

Решение

Задания

См. также

Примечания

Решение

```
#include <assert.h>
#include <stdio.h>

int
main ()
{
  int c;
  while ((c = getchar ()) != EOF) {
    printf (" %3d%s", c, (c == '\n' ? "\n" : ""));
  }
  assert (! ferror (stdin));

  return 0;
}
```

Главный цикл этой программы напоминает таковой для рассмотренной в разделе Максимум; в частности, мы вновь используем цикл «пока» (`while`).[1] Однако, вместо `scanf` для чтения чисел, здесь мы обращаемся к функции `getchar` для чтения отдельных *знаков* (кодов.)[2] Мы по-прежнему используем `printf` — для вывода кода символа в десятичной записи.[3]

С другой стороны, из *условия корректности* ввода исключается требование возврата именно `EOF` (как признака *исчерпания* входного потока), поскольку это условие *уже* является условием завершения *главного цикла*. Это различие связано с тем, что в данной программе не требуется опознавать «подходящий» ввод —

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

допустимой является совершенно *любая* последовательность символов (кодов.) Напротив, в предыдущей программе мы принимали исключительно целые числа в десятичной записи.

Требование *ложности* значения функции *признака ошибки* `ferror` для стандартного ввода (`stdin`) по завершению *главного цикла* остается в силе.[4]

Подчеркнем, что диапазон возвращаемых функцией `getchar` значений — это диапазон «символьного» типа `char` *плюс одно значение*, а именно — признак конца потока `EOF`.[2] Как следствие, иногда встречаемое в примерах кода чтение символа из потока непосредственно в переменную типа `char` *не вполне корректно* — для этих целей следует *всегда* использовать переменную типа `int`.

Обратите внимание, что сформировать условие «конец потока» при вводе с клавиатуры можно вводом (в зависимости от системы и предполагая настройки по-умолчанию) `Control-d` или `Control-z` (также обозначаются `C-d`, `^D`, `C-z`, `^Z`.)

Задания

1. Проверьте работу программы вводом строки `Hello!` В коде ASCII, выводом программы окажется `72 101 108 108 111 33 10` (где 10 — управляющий код разрыва строки.)
2. Введите строку `Привет!`. В зависимости от системы и ее настроек (так называемой *локали*), вывод может содержать 8 (при использовании *однобайтных* кириллических кодировок) или 14 кодов (UTF-8).
3. Изучите коды, формируемые такими клавишами и сочетаниями, как `ESC`, `F5`, `↑`, `Alt-x`, `Control-r`.
4. Исследуйте работу программы на «нетекстовых» потоках небольшого (до примерно 500 байт) объема — изображениях (`.png`, `.jpeg`), упакованных файлах (`.gz`, `.bz2`) и др. Попробуйте установить закономерности. (Указание: обратите внимание на значения первых пяти—десяти кодов.)
5. Предложенный вариант программы разрывает (`\n`) выходные строки по границам входных. Реализуйте также разрыв выходных строк по достижению определенного количества выведенных в одной строке кодов (например, каждые 16.)
6. Разработайте варианты программы, выводящие считанные коды в шестнадцатиричном и восьмеричном представлениях. (Указание: воспользуйтесь материалом раздела Скалярные типы.)
7. Напишите программу, которая печатает все символы и их ASCII-коды.
8. Попробуйте напечатать как символ число больше 255. Что получается?
9. Ознакомьтесь с описанием программы `od` в стандарте POSIX и в документации к пакету GNU Coreutils.[5][6]

См. также

- Кодирование текста
- ASCII
- http://asciitable.com/
- http://digteh.ru/proc/text.php — запись текстов (букв) двоичным кодом.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Примечания

1. *6.8.5.1 The while statement (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
2. *7.21.7.6 The getchar function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
3. *7.21.6.1 The fprintf function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
4. *7.21.10.3 The ferror function (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
5. *od – dump files in various formats (англ.) IEEE Std 1003.1-2013. The IEEE and The Open Group. Проверено 2015-03-13 г.*
6. *od invocation (англ.) GNU Coreutils. Free Software Foundation. Проверено 2015-03-13 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/ASCII-коды_символов&oldid=146501

Эта страница в последний раз была отредактирована 22 мая 2018 в 21:03.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Верхний регистр

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Дано
: *текст* — последовательность кодов ASCII — на стандартном вводе программы. Последовательность конечна, но ее длина заранее неизвестна.

Надо
: вывести на стандартный вывод исходный текст, заменив в нем строчные буквы латиницы (*a, …, z*) прописными (*A, …, Z*).

Указание
: воспользоваться особенностью кодовой таблицы ASCII: коды прописных букв отличаются от кодов соответствующих строчных на постоянную величину (а именно: 32.)

Дополнительно
: предполагая поддержку средой русскоязычной *локали*, реализуйте аналогичное преобразование для кириллических текстов (используя стандартные функции языка.)

Содержание

Решение

Использование стандартной библиотеки

Кроме латиницы

Задания

См. также

Примечания

Решение

В следующей программе введенный пользователем текст на латинице переводится в верхний регистр, то есть строчные буквы становятся заглавными. Все прочие коды — включая заглавные буквы и символы, не являющиеся латинскими буквами — не меняются.

Обратите внимание на то, что с символами (переменными типа `char`) можно оперировать как с числами. В частности, `'Z' - 'A'` есть число, равное разности кодов символов A и Z, то есть (при использовании кодовой таблицы ASCII) 25 — число букв в латинском алфавите минус 1. Символы можно также сравнивать друг с другом, при этом сравниваются их коды.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

```c
#include <assert.h>
#include <stdio.h>

int
main ()
{
  int c;
  while ((c = getchar ()) != EOF) {
    if ('a' <= c && c <= 'z') {
      c += 'A' - 'a';
    }
    putchar (c);
  }
  assert (! ferror (stdin));

  return 0;
}
```

Главный цикл (и *условие корректности* ввода) этой программы почти полностью совпадают с рассмотренными в разделе ASCII коды символов. Однако, вместо использования `printf` для вывода *значений* считанных кодов, здесь мы используем функцию `putchar` для вывода *самих* кодов (знаков), возможно — после преобразования.[1] В коде по-прежнему используются функции `getchar` и `ferror`.[2][3]

Обратите внимание, что сформировать условие «конец потока» при вводе с клавиатуры можно вводом (в зависимости от системы и предполагая настройки по-умолчанию) `Control-d` или `Control-z` (также обозначаются `C-d`, `^D`, `C-z`, `^Z`.)

Использование стандартной библиотеки

Приведенный выше код можно еще более упростить, если воспользоваться стандартной функцией преобразования регистра `toupper`, *объявленной* в заголовке `ctype.h`.[4]

Некоторые *реализации* языка, кроме того, определяют функции `strlwr` и `strupr`, которые переводят в нижний и верхний регистр *строки*, а не отдельные символы. Эти функции, однако, не являются стандартными и их наличие (или отсутствие) зависит от выбранной реализации языка.

```c
#include <assert.h>
#include <ctype.h>
#include <stdio.h>

int
main ()
{
  int c;
  while ((c = getchar ()) != EOF) {
    putchar (toupper (c));
  }
  assert (! ferror (stdin));

  return 0;
}
```

Кроме латиницы

Стандарт предусматривает также ряд средств для поддержки письменностей, отличных от «базовой» (26-символьной) латиницы. В основе такой поддержки — два следующих понятия:

1. *локаль* — набор сведений о письменности, языке и культуре, включающий, помимо прочего, таблицы соответствия символов верхнего и нижнего регистров;
2. *широкие символьные типы* (`wchar_t`, `wint_t`; англ. wide character) — для представления ряда письменностей (например — основанных на иероглифах), а равно для представления текстов на нескольких языках, «байтового» типа `char` оказывается недостаточно; как следствие, в задачах, связанных с обработкой *многоязычных* текстов, его заменяет тип `wchar_t`; аналогично, в операциях ввода-вывода используется тип `wint_t` (и константа `WEOF`.)

Соответственно, в исходный вариант вносятся следующие изменения.

1. Появляется вызов `setlocale (LC_ALL, "")`, *инициализирующий* локаль в соответствии с текущими настройками. (По-умолчанию, при запуске программы полагается `setlocale (LC_ALL, "C")` — установка «стандартной» локали.)[5]
2. Функции `getwchar`, `towupper`, `putwchar` заменяют соответствующие для «узких» символов (`getchar`, `toupper`, `putchar`.) Соответственно меняются тип (`int` → `wint_t`) и константа признака исчерпания входного потока (`EOF` → `WEOF`.)[6][7][8]
3. Включаются соответствующие *заголовки* (`locale.h`, `wchar.h`, `wctype.h`.)

```c
#include <assert.h>
#include <locale.h>
#include <stdio.h>
#include <wchar.h>
#include <wctype.h>

int
main ()
{
  char *rl
    = setlocale (LC_ALL, "");
  assert (rl != 0);

  wint_t c;
  while ((c = getwchar ()) != WEOF) {
    putwchar (towupper (c));
  }
  assert (! ferror (stdin));

  return 0;
}
```

Отметим, впрочем, что принятые в конкретном языке правила обращения со строчными и заглавными могут оказаться сложнее простой зависимости символ—символ, *предполагаемой* в данном примере. Так, например, правила немецкого языка ставят в соответствие строчной «ß» *пару* заглавных — «SS» (например: Straße → STRASSE), что ни коим образом не может быть реализовано функцией `towupper`.

Задания

1. Проверьте работу программы вводом строк `Hello!` и `Привет!`. Удостоверьтесь, что «базовые» варианты решения выполняют заданное преобразование лишь для для первой из них, в то время как «интернационализованное» решение действует и в отношении кириллического ввода.
2. Разработайте варианты программы, выполняющие (вновь для латиницы) a. преобразование *верхнего* регистра в нижний и b. «переключение» регистра — `Hello!` → `hELLO!`. (Указание: воспользуйтесь стандартными функциями `islower` и `isupper`.[9][10])

См. также

- ASCII коды символов
- ASCII
- UTF-8
- Заглавная буква эсцет

Примечания

1. *7.21.7.8 The putchar function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
2. *7.21.7.6 The getchar function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
3. *7.21.10.3 The ferror function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
4. *7.4.2 Character case mapping functions* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
5. *7.11.1.1 The setlocale function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
6. *7.29.3.7 The getwchar function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
7. *7.30.3.1.2 The towupper function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
8. *7.29.3.9 The putwchar function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
9. *7.4.1.7 The islower function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
10. *7.4.1.11 The isupper function* *(англ.)* *WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Верхний_регистр&oldid=106991

Эта страница в последний раз была отредактирована 14 марта 2015 в 12:12.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Скобочки

Задача "Скобки"

Программа "Скобки" должна определять правильность введённой скобочной структуры. Правильная скобочная структура, это то, что можно получить из арифметического выражения со скобками, выкинув все знаки операций и цифры. Например:

- Правильные скобочные структуры
  - ()
  - ()()
  - ((()))
  - (()())
  - ((()())())
- Неправильные скобочные структуры
  -)(
  - ())(
  - (
  - ())
  -)(())(

Вход: Слово в алфавите из двух круглых скобочек (и). Длина слова меньше 100000 символов.

Выход: Либо NO, либо YES.

```c
#include <stdio.h>

int main(void)
{
    int ch, a = 0;
    printf("Введите слово из скобок: ");

    do {
        ch = getchar();
        if (ch == '(')
            a++;
        else if (ch == ')' && --a < 0)
            break;
    } while (ch != '\n');

    if (a == 0)
        printf("Правильно\n");
    else
        printf("Не правильно\n");

    return 0;
}
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в

формате ADX

Здесь нам встречается оператор `==`, который соответствует логическому "тождественно равно". Когда мы пишем `i = 0`, то мы присваиваем переменной `i` значение 0.

Выражения a++ и --a меняют значения в памяти напрямую, то есть выражение --a в примере уменьшит значение переменной a вне зависимости от того, истинно ли выражение (--a < 0), или ложно.

Выражение `a == 0` является проверкой равенства, её значение равно "истина" или "ложь".

В языке C/C++ "истина" соответствует любому ненулевому числу. Поэтому, вместо `a == 0` мы могли бы написать равносильное `a`, но это не рекомендуется делать.

Обратите внимание на то, что программа не хранит скобочную структуру в памяти. В этом нет необходимости, поскольку алгоритм **однопроходный**, то есть достаточно конечной памяти и одного пробега по сколь угодно большим входным данным, чтобы получить результат.

Свойством однопроходности обладает также рассмотренный нами алгоритм поиска максимума из чисел.

Задача сортировка чисел не является однопроходным алгоритмом.

1. Какие из перечисленных задач имеют однопроходные решения:
   - Найдите среднее арифметическое чисел.
   - Найдите дисперсию чисел (средний квадрат отклонения от среднего).
   - Найдите три самых маленьких числа среди введенных.
2. Напишите программу, которая проверяет правильность скобочной структуры, составленной из нескольких типов скобок (круглых, квадратных и фигурных).Например, `({()[]})` — правильная структура, а `{()[)}]` — неправильная.
3. Напишите программу, которая выводит все правильные скобочные структуры заданной длины. А именно, опишите рекурсивную функцию "вывести все правильные скобочные структуры длины n".
4. Напишите программу, которая определяет число правильных скобочных структур заданной длины. Сведите задачу к множеству задач: число префиксов правильных скобочных структур длины n, которые имеют k незакрытых скобок.

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Скобочки&oldid=93318

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:42.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Факториал

Факториалом числа *n* называют произведение первых *n* натуральных чисел:

$$n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{i=1}^{n} i.$$

Ниже мы рассмотрим примеры программ, содержащих вычисляющую факториал заданного числа функцию `factorial`.

Содержание

Реализация рекуррентной формулы

Использование хвостовой рекурсии

Явно итеративный вариант

Вариант «произвольный»

Задания

См. также

Примечания

Реализация рекуррентной формулы

```c
#include <stdio.h>

static int
factorial (int n)
{
  return (n < 2) ? 1 : n * factorial (n - 1);
}

int
main (void)
{
  int n;
  while (scanf ("%d", &n) == 1) {
    printf ("%d\n", factorial (n));
  }
  return 0;
}
```

Определение функции `factorial` выше основано на следующей *рекуррентной* формуле:

$$n! = n \cdot (n-1)!, \quad 0! = 1.$$

Для вычисления факториала *n*! эта функция вызывает саму себя с аргументом *n* − 1.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

Тернарный оператор (`? :`) в выражении `(n < 2) ? 1 : n * factorial (n - 1)` для возвращаемого значения (`return`) «обрывает» рекурсию, когда аргумент функции становится меньшим 2. В противном случае, функция `factorial` постоянно бы вызывала саму себя, и во время выполнения программы мы получили бы сообщение об ошибке «переполнение стека» или «превышена глубина рекурсии». Для того, чтобы этого не было, необходимо, чтобы при некоторых значения аргумента, функция не вызывала саму себя, а вычисляла свое значение «самостоятельно».

Идея рекурсии заключается в сведении задачи к этой же задаче, но для более простых случаев (например, для меньшего значения аргумента *n*). Процесс сведения задачи к предыдущей должен когда-нибудь заканчиваться, поэтому рекурсивные функции для простейших входных данных должны «знать», чему они равны и не делать рекурсивных вызовов.

Использование хвостовой рекурсии

Возможность рекурсии ограничена предельной глубиной *стека возврата*. В некоторых случаях, однако, реализация языка программирования может самостоятельно свести рекурсию к итерации.

В примере ниже, мы воспользуемся хвостовой рекурсией, сведение которой к итерации хотя и не требуется стандартом, но все же реализуется, например, компилятором C, входящим в распространенный комплект GCC. Для этого, нам потребуется определить вспомогательную функцию `factorial_1`, вычисляющую значение *p n*!, где *p*, *n* — аргументы функции.

Обратите внимание, что в этом примере мы не приводим ни определения функции `main`, ни *директив препроцессора* `#include` подключения используемых нами *заголовков* — их следует заимствовать из предыдущего примера.

```
static int
factorial_1 (int n, int p)
{
  return (n < 2) ? p : factorial_1 (n - 1, p * n);
}

static int
factorial (int n)
{
  return factorial_1 (n, 1);
}
```

Явно итеративный вариант

Наконец, в варианте ниже мы вовсе исключим рекурсию из определения функции `factorial`, явно сведя ее к итерации.

```
static int
factorial (int n)
{
  int r;
  for (r = 1; n > 1; r *= (n--))
    ;
  return r;
}
```

В отличие от первого варианта (но аналогично варианту с хвостовой рекурсией) в данном примере нам потребовалась дополнительная локальная переменная. В случае «нехвостовой» рекурсии — роль такой переменной фактически играет растущий *стек возврата*. Отметим, впрочем, что в данной конкретной задаче это не столь важно.

Вариант «произвольный»

Стандарт требует поддержки реализацией числовых типов разрядности 8, 16, 32 и 64 бит.[1] Несложно убедиться, однако, что уже 21! = 51 090 942 171 709 440 000 — что превышает предельные значения для 64-битового числа — как со знаком (9 223 372 036 854 775 807), так и без знака (18 446 744 073 709 551 615).

Если по каким-либо причинам требуется вычислить *точные* значения факториала для чисел от 21 и выше, следует использовать библиотеки операций с числами *произвольной* разрядности — как, например, GNU MP.[2]

```c
#include <stdio.h>
#include <gmp.h>

static void
factorial (long n, mpz_t r)
{
  mpz_init_set_si (r, 1);
  for (; n > 1; n--) {
    mpz_mul_si (r, r, n);
  }
}

int
main (void)
{
  int n;
  mpz_t r;
  while (scanf ("%d", &n) == 1) {
    factorial (n, r);
    gmp_printf ("%Zd\n", r);
  }
  return 0;
}
```

Задания

1. Улучшите программы, добавив к ним диагностику ошибки во входных данных: если $n < 0$, то программа должна ясно сообщать о неопределенности факториала для отрицательных чисел.
2. Опытным путем определите максимальные значения аргумента и результата для «стандартных» программ выше. Исследуйте влияние на эти значения замены используемого в программах числового типа `int` на `short`, `long` и `long long`. **Обратите внимание**, что при этом также потребуется соответственно изменить *строку формата* (первый аргумент) функции `printf`.
3. Определите также максимальные значения аргумента и результата для этих программ при использовании чисел с плавающей запятой — числовых типов `double`, `float`, `long double`.

См. также

- Реализации алгоритмов/Факториал (на различных языках)

Примечания

1. *7.20.1.2 Minimum-width integer types (англ.) WG14 N1570 Committee Draft. ISO/IEC (2011-04-12). Проверено 2012-11-19 г.*
2. *The GNU MP Bignum Library (англ.) Проверено 2015-04-04 г.*

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Факториал&oldid=109819

Эта страница в последний раз была отредактирована 4 апреля 2015 в 17:10.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Степень числа

Материал из Викиучебника — открытых книг для открытого мира

< Язык Си в примерах

Для вычисления функции $f(n) = a^n$ есть рекуррентная формула:

$$a^n = a \cdot a^{n-1}, \quad a^0 = 1.$$

Вот программа, которая основана на этой формуле:

```
/*
  Степень числа: простая рекурсия
*/

#include<stdio.h>

double power(double x, long n) {
   if(n == 0) return 1;
   if(n < 0) return power ( 1.0 / x, -n);
   return x * power(x, n - 1);
}

void main() {
    double x;
    long n;
    while (scanf ("%lf %ld", &x, &n) == 2) {
       printf("%lf\n", power (x, n));
    }
}
```

Приведёный выше пример не будет работать для отрицательных показателей степени (см. третью строку функции "power"). Правильнее было бы так:

```
/*
  Степень числа: простая рекурсия
*/

#include<stdio.h>

double power(double x, long n) {
   if(n == 0) return 1.0;
   if(n < 0) return 1.0 / (x * power (1.0 / x, n + 1));
   return x * power(x, n - 1);
}

void main() {
    double x;
    long n;
    while (scanf ("%lf %ld", &x, &n) == 2) {
       printf("%16.16lf\n", power (x, n));
    }
}
```

Но есть более «умная» рекурсивная функция:

$$a^n = \begin{cases} a \cdots a^{n-1}, & \text{if } n \text{ is odd} \\ (a^{n/2})^2, & \text{if } n \text{ is even} \end{cases}$$

Например, если обозначить стрелочкой $\rightarrow$ слово «сводится к », то при вычислении a^{12} для первой рекурсии получим цепочку длины 12:

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

$a^{12} \to a^{11} \to a^{10} \to a^9 \to a^8 \to a^7 \to a^6 \to a^5 \to a^4 \to a^3 \to a^2 \to a^1 \to a^0$.

А для второй рекурсии цепочку из 5 шагов: $a^{12} \to a^6 \to a^3 \to a^2 \to a^1 \to a^0$.

Для больших n разница в длине цепочки более разительная. В частности a^{10000} первой рекурсией вычисляется за 10000 шагов, а второй -- за 19 шагов.

```
/*
 Программа 2: степень числа -- оптимизированная рекурсия.
*/
double power(double x, long n)
{
    double tmp;
   if(n == 0) return 1;
   if(n < 0) return power ( 1 / x, -n);
   if(n % 2) return x * power (x, n - 1);
   return power(x * x, n / 2);
}
```

```
/*
   Программа 3: степень числа -- оптимизированный алгоритм без рекурсии.
*/
double power(double x, long n)
{
    double a = 1;
    while(n) {
        if(n % 2) {
            a *= x;
            n--;
        }
        else{
            x *= x;
            n /= 2;
        }
    }
    return a;
}
```

- Напишите программу, вычисляющую double в степени double.
- Сколько шагов требуется для вычисления a^{30} вторым методом?
- Покажите, что второй алгоритм выполняется за логарифмическое по n число шагов, а точнее ограничено сверху $2 \cdot \log_2 n$ (еще точнее: в точности равно числу знаков в двоичной записи числа n плюс число единичек в этой записи).
- Объясните, как работает программа 3.

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Степень_числа&oldid=136595

Эта страница в последний раз была отредактирована 25 марта 2017 в 20:53.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Треугольник Паскаля

Материал из Викиучебника — открытых книг для открытого мира

Всем известны простые формулы

- $(a+b)^2 = a^2 + 2ab + b^2$
- $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$

А как находить коэффициенты в разложении $(a+b)^n$?

Рассмотрим следующую таблицу:

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

$(a+b)^0$	$=$	1	0						1					
$(a+b)^1$	$=$	$a+b$	1					1		1				
$(a+b)^2$	$=$	$a^2+2ab+b^2$	2				1		2		1			
$(a+b)^3$	$=$	$a^3+3a^2b+3ab^2+b^3$	3			1		3		3		1		
$(a+b)^4$	$=$	$a^4+4a^3b+6a^2b^2+4ab^3+b^4$	4		1		4		6		4		1	
$(a+b)^5$	$=$	$\dots$	5	1		5		10		10		5		1
$(a+b)^6$	$=$	$\dots$	6	1	6		15		20		15		6	1

То, что нарисовано справа, называется *треугольником Паскаля* — в n-й строчке этого треугольника находятся коэффициенты для разложения $(a+b)^n$.

Число номер $k+1$ в n-й строчке называется биномиальным коэффициентом C_n^k.

Например, $C_3^0 = 1, \quad C_5^2 = 10, \quad C_4^2 = 6.$

Стороны треугольника Паскаля состоят из единичек. Каждое число внутри треугольника Паскаля равно сумме двух чисел, стоящих над ним справа и слева в предыдущей строчке:

$$C_{n+1}^{k+1} = C_n^{k+1} + C_n^k.$$

Эти числа возникают в задаче о числе сочетаний: C_n^k — это число способов выбрать k элементов из n различных. Например, число байт, в которых ровно 3 единицы — это число C_8^3 — число способов выбрать три бита, в которых будут стоять единицы, из восьми бит байта.

Докажите, что

$C_n^1 = n, \quad C_n^k + C_n^{k+1} = C_{n+1}^{k+1}, \quad C_n^k = \dfrac{n!}{k!(n-k)!}.$

Рассмотрим две программы, которые решают следующие задачи:

1. Запрограммировать функцию $C(n,k) = C_n^k$.
2. Вывести на экран n строчек треугольника Паскаля.

```c
/*
   Вычисление биномиальных коэффициентов.
 */
#include <stdio.h>

long
C (long n, long k)
{
  if (k == 0 || n == k)
    return 1;
  return C (n - 1, k - 1) + C (n - 1, k);
}

int
main ()
{
  long n, k;
  scanf ("%ld%ld", &n, &k);
  printf ("%ld ", C (n, k));
  return 0;
}
```

- Сколько раз вызовется функция C(., .) при вычислении C(n, k)?
- Докажите, что время вычисления C_n^k по приведенному алгоритму пропорционально C_n^k.
- Оцените ассимптотику $C_n^{n/2}$, а именно, напишите программу, которая вычисляет $\log C_n^{n/2}$ для $n = 2, 4, \ldots, 40$ и нарисуйте график

$\log C_n^{n/2}$ от n.

```
/*
    Вычисление n-й строки треугольника Паскаля.
*/
#include <stdio.h>
#define N 1000

int
main ()
{
  long c[N];
  long n, i, j;
  scanf ("%ld", &n);
  for (i = 1; i <= n; i++)
    c[i] = 0;
  c[0] = 1;
  for (j = 1; j <= n; j++)
    for (i = j; i >= 1; i--)
      c[i] = c[i - 1] + c[i];
  for (i = 0; i <= n; i++)
    printf ("%ld ", c[i]);
  return 0;
}
```

- Докажите, что указанный алгоритм вычисления *n*-й строчки треугольника Паскаля

работает быстрее, чем алгоритм вычисления C_n^k из предыдущей программы, а именно время работы пропорционально n^2.

- Начиная с какого *n* самое большое число из *n*-й строчки треугольника Паскаля не умещается в тип `long`?

Еще один вариант реализации вычисления Биномиального коэффициента из общей формулы C_n^k (без рекурсии):

```
/*
  Вычисление биномиальных коэффициентов
*/
#include <assert.h>
#include <stdio.h>

int
binomial (int row, int pos)
{
  int koef = 1;
  int i;
  for (i = pos + 1; i <= row; i++)
    koef = koef * i;
  for (i = 1; i < (row - pos + 1); i++)
    koef = koef / i;
  return koef;
}

int
main (void)
{
  int row, pos;
  int r = scanf ("%i%i", &row, &pos);
  assert (r == 2);
  printf ("%i", binomial (row, pos));
  return 0;
}
```

Если посмотреть на Треугольник Паскаля, то можно заметить симметрию. Если исходить из нее и общей формулы C_n^k, то можно еще немного улучшить алгоритм (уменьшив число проходов) из кода выше сокращая числитель на больший элемент знаменателя:

```c
/*
  Вычисление биномиальных коэффициентов
*/
#include <assert.h>
#include <stdio.h>

int
binomial (int row, int pos)
{
  int koef = 1;
  int i;
  if (row - pos > pos)
    pos = row - pos;
  for (i = pos + 1; i <= row; i++)
    koef = koef * i;
  for (i = 1; i < (row - pos + 1); i++)
    koef = koef / i;
  return koef;
}

int
main (void)
{
  int row, pos;
  int r = scanf ("%i%i", &row, &pos);
  assert (r == 2);
  printf ("%i", binomial (row, pos));
  return 0;
}
```

Стоит учитывать, что в реализованных примерах нет защиты от дурака (переполнение значения переменной).

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Треугольник_Паскаля&oldid=89900

Эта страница в последний раз была отредактирована 13 апреля 2014 в 10:41.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Корень уравнения

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Задача: Рассмотрим трансцендентное уравнение, которое не имеет явной формулы для корня: $e^x = x + 2$. С помощью компьютера необходимо найти положительный корень этого уравнения с погрешностью по оси x не более 10^{-10}.

Вот решение этой задачи:

```
/* root.c: Вычисление корня трансцендентного уравнения */
#include <stdio.h>
#include <math.h>
#define EPS 1e-10             // точность результата
double f(double x) {
    return exp(x) - 2 - x;
}
int main() {
   double l = 0, r = 2, c;
   while( r - l > EPS ) {
      c = ( l + r ) / 2;      // вычисляем середину отрезка;
      if( f(c) * f(r) < 0 )   // узнаем, в какой из частей
          l = c;              // находится искомый корень;
      else
          r = c;
   }
   printf ("%.10lf\n", (l + r)/2 ); // выводим результат
}
```

При компиляции этой программы с помощью GCC следует указать опцию `-lm`, которая указывает что при компоновке программы необходимо подключить библиотеку `libm` с математическими функциями:

```
bash$ gcc -lm root.c -o root
```

В библиотеке `mlib`, в частности, определена функция `exp`, вычисляющая экспоненту, а также многие другие математические функции: тригонометричекие (`sin`, `cos`, `asin`, `acos`, `tan`, ...), корень (`sqrt`), степень (`pow`), логарифм (`log`), ...

Директива `#define EPS 1e-10` означает: в тексте программы идентификатор EPS заменить на число `1e-10`, то есть $1 \cdot 10^{-10}$. Число `EPS` — это погрешность по оси x, с которой мы хотим найти корень.

Алгоритм вычисления корня основан на методе деления пополам: Предположим, что искомый корень находится между `l = 0` и `r = 2`. Найдем середину `c` отрезка `[l, r]`. Корень находится на одном из отрезков: либо на `[l, c]`, либо на `[c, r]`, а именно, на том, значение функции на концах которого имеет разные знаки (вспомните теорему Ролля про непрерывную функцию из курса мат. анализа). Выберем нужный из двух отрезков и применим к

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

нему такие же рассуждения. Будем осуществлять деление пополам, пока размер отрезка не станет меньше необходимой точности. (В методе деления отрезка пополам отрезок на оси x делится пополам пока $|f(x_c)| > \varepsilon_f$, в приведённом же методе отрезок на оси x может достичь заданной величины ε_x, а значения функций $f(x)$ (особенно крутых) на оси y могут очень далеко отстоять от нуля, при пологих же функциях $f(x)$ этот метод приводит к большому числу лишних вычислений.)

Вопросы и задачи

- За один шаг длина отрезка `[l, r]` уменьшается в два раза. Сколько нужно шагов, чтобы уменьшить отрезок более, чем в 1000 раз?
- Сколько требуется шагов, чтобы начиная с отрезка длины $r - l = 2$ дойти до отрезка длины меньше 10^{-10}? Сколько требуется шагов, чтобы найти корень с точностью до 100 знаков после запятой?
- В случае деления пополам у нас есть нижняя и верхняя граница для значения корня. С каждым шагом эти границы сближаются. В методе Ньютона нахождения корня уравнения у нас имеется одно число *x* — текущее приближение корня. И следующее приближение получается по следующему алгоритму: находим точку на графике с абсциссой x и проводим из неё касательную к графику; абсцисса точки пересечения касательной с осью абсцисс будет новым значением *x*. Так делается до тех пор, пока новое *x* отличается от старого на число меньше, чем $10^{-10}/2$. Реализуйте этот алгоритм. Для этого вам понадобится определить еще одну функцию, которая возвращает значение производной $f'(x) = (e^x - x - 2)' = e^x - 1$.

Универсальная функция вычисления корня

В рассмотренной программе вычисляется нуль вполне конкретной функции: f(x) = exp(x) - 2 - x. Следуя идеологии Code reuse (повторное использование кода), полезно сделать функцию, которая умеет находить нули произвольных функций. Для этого нужно научиться передавать функцию в качестве аргумента --- это возможно, и совсем несложно:

```c
/* Универсальная функция вычисления корня уравнения f(x) = 0  */
#include <stdio.h>
#include <math.h>
#define EPS 1e-16
double root(double l, double r, double (*f)(double)) {
   double c;
   while( r - l > EPS ) {
      c = ( l + r ) / 2;
      if( f(c) * f(r) < 0 )
          l = c;
      else
          r = c;
   }
   return l;
}
double f1(double x) {return cos(x) - 3 * x; }
double f2(double x) {return exp(x) - x - 2; }
int main() {
    printf("root1 = %lf\n", root(0, 2, f1)); // вычисляем и выводим корень уравнения f1(x) = 0
    printf("root2 = %lf\n", root(0, 2, f2)); // вычисляем и выводим корень уравнения f2(x) = 0
    return 0;
}
```

Вывод программы выглядит следующим образом:

```
root1 = 0.316751
root2 = 1.146193
```

См. также

- Библиотека complex

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Корень_уравнения&oldid=93335

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:51.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Система счисления

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Число 235 в десятичной системе счисления есть

$$235_{10} = 2 \cdot 10^2 + 3 \cdot 10 + 5$$

Число 235 в восьмеричной системе счисления есть

$$235_8 = 2 \cdot 8^2 + 3 \cdot 8 + 5$$

Пусть Q — натуральное число. Тогда представить число N в Q-ичной системе счисления означает представить число N в виде суммы различных степеней Q с целыми коэффициентами из диапазона $[0; Q-1]$:

$$N = a_0 \cdot Q^0 + a_1 \cdot Q^1 + a_2 \cdot Q^2 + \ldots$$

Q-ичная запись числа N — это набор коэффициентов $(a_m, a_{m-1}, \ldots, a_0)$,где a_m — последний ненулевой коэффициент.

Вопросы

1. Докажите методом математической индукции, что это разложение существует и единственно. Для тренировки разберите отдельно случай $Q = 2$: любое натуральное число представляется в виде суммы различных степеней двойки и притом единственным образом.

Примеры:

- $1_{10} = 1_2$
- $2_{10} = 0 \cdot 2^0 + 1 \cdot 2^1 = 10_2$
- $3_{10} = 1 \cdot 2^0 + 1 \cdot 2^1 = 11_2$
- $4_{10} = 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 = 100_2$
- $5_{10} = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 = 101_2$
- $15_{10} = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 = 1111_2$
- $16_{10} = 0 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 10000_2$

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

- $255_{10} = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7 = 11111111_2$

2. Что делает приведённая ниже программа?

```
#include <stdio.h>
void main ()
{
    int n;
    scanf ("%d", &n);
    while(n)
    {
        printf("%d", n%2);
        n /= 2;
    }
}
```

3. Напишите программу, которая введённое натуральное (число в десятичной записи) переводит в восьмеричную систему счисления.

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Система_счисления&oldid=93323

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:43.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Сортировка

Задача «сортировки» (упорядочения) — одна из первых интересных и сложных задач теории алгоритмов. Общие принципы освещает статья «Алгоритмы сортировки», здесь же мы рассматриваем способы упорядочения посредством языка Си.

Содержание

Метод «пузырька»

Функция qsort из библиотеки stdlib

Динамическое выделение памяти

Программа упорядочения строк в алфавитном порядке

Примечания

Метод «пузырька»

Один из простейших алгоритмов решения — метод «пузырька».

```
#include <stdio.h>
#define N 1000
int main() {
   int n, i, j;
    scanf_s("%d", &n);
   int a[n];
   // считываем количество чисел n

   // формируем массив n чисел
   for(i = 0 ; i < n; i++) {
       scanf_s("%d", &a[i]);
   }
   for(i = 0 ; i < n - 1; i++) {
      // сравниваем два соседних элемента.
      for(j = 0 ; j < n - i - 1 ; j++) {
          if(a[j] > a[j+1]) {
             // если они идут в неправильном порядке, то
             //  меняем их местами.
             int tmp = a[j];
             a[j] = a[j+1] ;
             a[j+1] = tmp;
          }
       }
   }
}
```

Понятно, что после первого «пробега» самый большой элемент массива окажется на последнем месте. После второго пробега мы будем уверены, что второй по величине элемент находится на предпоследнем месте.

Задача: Докажите, что достаточно $n-1$ пробега, чтобы элементы массива упорядочились.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

Решив эту задачу, вы докажете, что метод «пузырька» решает задачу сортировки.

Функция `qsort` из библиотеки `stdlib`

Два оператора `for`, в которых происходит сортировка, можно заменить на одну строку:

```
qsort(a, n, sizeof(int), cmp );
```

Это функция, описанная в стандартной библиотеке ANSI C и объявлена в заголовочном файле `stdlib.h`.

Поэтому в начале программы нужно добавить

```
#include <stdlib.h>
```

Функцией `qsort` можно упорядочивать объекты любой природы. По сути, она предназначена упорядочивать множества блоков байтов равной длины. Второй аргумент функции — это число таких блоков, третий аргумент — длина каждого блока. Первый аргумент — это адрес, где находится начало первого блока (предполагается, что блоки в памяти расположены друг за другом подряд).

Четвёртый аргумент функции `qsort` — это имя функции, которая умеет сравнивать два элемента массива. В нашем случае это

```
int cmp(const void *a, const void *b) {
     return *(int*)a - *(int*)b;
 }
```

В силу указанной универсальности функции сортировки, функция сравнения получает в качества аргумента адреса двух блоков, которые нужно сравнить и возвращает 1, 0 или -1:

> положительное значение, если a > b
> 0, если a == b
> отрицательное значение, если a < b

Поскольку у нас блоки байт -- это целые числа (в 32-битной архитектуре это четырёхбайтовые блоки), то необходимо привести данные указатели типа `(const void*)` к типу `(int *)` и осуществляется это с помощью дописывания перед указателем выражения «(const int*)». Затем нужно получить значение переменной типа int, которая лежит по этому адресу. Это делается с помощью дописывания спереди звездочки.

Таким образом, мы получили следующую программу

```
 #include <stdio.h>
 #include <stdlib.h>
 #define N 1000
 int cmp(const void *a, const void *b) {
     return *(int*)a - *(int*)b;
 }
 int main() {
    int n, i,j;
    int a[N];
```

```
    scanf("%d", &n);
    for(i = 0 ; i < n; i++) { // ЧИТАЕМ ВХОД
        scanf("%d", &a[i]);
    }
    qsort(a, n, sizeof(int), cmp ); // СОРТИРУЕМ
    for(i = 0 ; i < n; i++) { // ВЫВОДИМ РЕЗУЛЬТАТ
        printf("%d ", a[i]);
    }
    return 0;
}
```

Динамическое выделение памяти

Ниже приведена программа, где память под массив выделяется динамически:

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
int cmp(const void *a, const void *b) {
    return *(int*)a - *(int*)b;
}
int main() {
   int n, i;
   int *a;
   scanf("%d", &n);
   a = (int*) malloc(sizeof(int)*n);
   for(i = 0 ; i < n; i++) {
       scanf("%d", &a[i]);
   }
   qsort(a, n, sizeof(int), cmp );
   for(i = 0 ; i < n; i++) {
       printf("%d ", a[i]);
   }
   free(a);
   return 0;
}
```

Функция malloc (от англ. *memory allocation --- выделение памяти*) делает запрос к ядру операционной системы по выделению заданного количества байт. Единственный аргумент этой функции — число байт, которое вам нужно. В качестве результата функция возвращает указатель на начало выделенной памяти. Указатель на начало выделенной памяти &mbsah — это адрес ячейки памяти, начиная с которого идут N байт, которые вы можете использовать под любые свои нужды. Всю память, которая была выделена с помощью функции malloc, нужно освобождать с помощью функции free. Аргумент функции free — это указатель на начало выделенной когда-то памяти.

Программа упорядочения строк в алфавитном порядке

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#define N 100
#define M 30
int main(int argc, char* argv[]) {
   char a[N][M];
   int n, i;
   scanf("%d", &n);
   for (i=0; i<n; i++)
      scanf("%s", &a[i]);
   qsort(a, n, sizeof(char[M]), (int (*)(const void *,const  void *)) strcmp);
```

```
    for (i=0; i<n; i++)
       printf("%s\n", a[i]);
    return 0;
 }
```

Обратите внимание на сложное приведение типов.

Функция strcmp, объявленная в файле string.h имеет следующий прототип:

```
int strcmp(const char*, const char*);
```

То есть функция получает два аргумента -- указатели на кусочки памяти, где хранятся элементы типа char, то есть два массива символов, которые не могут быть изменены внутри функции strcmp (запрет на изменение задается с помощью модификатора const)[1] (https://ru.wikibooks.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%A1%D0%B8_%D0%B2_%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D0%B0%D1%85/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0#endnote_strcmp).

В то же время в качестве четвертого элемента функция qsort хотела бы иметь функцию типа

```
int cmp(const void*, const void*);
```

В языке Си можно осуществлять приведение типов являющихся типами функции. В данном примере тип

```
int (*)(const char*, const char*); // функция, получающая два элемента типа 'const char *' и возвращающая элемент типа 'int'
```

приводится к типу

```
int (*)(const void*, const void*); // функция, получающая два элемента типа 'const void *' и возвращающая элемент типа 'int'
```

Примечания

1. ^ Функция strcmp в соответствии с описанием, выдаваемым командой man 3 strcmp, осуществляет сравнение двух строк и определяет, какая из двух строк идёт первой в алфавитном порядке (сравнивает две строки в лексикографическом порядке), а именно: она возвращает значение больше нуля, если первая строка "больше" второй (идёт после второй в алфавитном порядке), 0 – если они совпадают, и значение меньше нуля– если первая строка "меньше" второй.

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Сортировка&oldid=158774

Эта страница в последний раз была отредактирована 18 ноября 2019 в 18:51.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Библиотека complex

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Содержание

Создание библиотек. Библиотека для работы с комплексными числами.

- Создание библиотеки
- Пример использования библиотеки
- Создание и использование библиотек в GNU C
  - Подключение библиотеки в виде исходников
  - Подключение библиотеки в виде динамически загружаемой библиотеки
  - Подключение библиотеки в виде статически загружаемой библиотеки
  - Задачи

Создание библиотек. Библиотека для работы с комплексными числами.

Создание библиотеки

Библиотека у нас будет иметь два исходных файла: `complex.h` и `complex.c`

```
/*
   complex.h
*/
typedef struct
   {
    double a,b;
   } complex_t;
complex_t mul(complex_t x, complex_t y);
```

```
/*
   complex.c
*/
#include "complex.h"
complex_t mul(complex_t x, complex_t y)
{
    complex_t t;
    t.a = x.a * y.a - x.b * y.b;
    t.b = x.a * y.b + x.b * y.a;
    return t;
}
```

Пример использования библиотеки

Теперь несложно использовать эту библиотеку.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

Для этого нужно:

- добавить в заголовок c/cpp-файла строчку `#include "complex.h"` ;
- сделать так, чтобы компилятор находил `complex.h` ;
- сделать так, чтобы сборщик (linker) находил файл библиотеки `complex` .

```c
/*
   test_complex.c
*/
#include <stdio.h>
#include "complex.h"
int main()
{
    complex_t x = {1,2};
    complex_t y = {3,4};
    complex_t z = mul(x,y);
    printf ("z = (%lf, %lf)\n", z.a, z.b );
    return 0;
}
```

Создание и использование библиотек в GNU C

Подключение библиотеки в виде исходников

Есть простой способ подключения библиотеки функций прямо в виде исходников:

```
$ gcc test_complex.c complex.c -o test_complex
```

В итоге получается запускаемый файл `test_complex`.

Подключение библиотеки в виде динамически загружаемой библиотеки

```
$ gcc -shared complex.c -o libcomplex.so           # создание динамически загружаемой библиотеки
$ gcc test_complex.c -L. -lcomplex -o test_complex # компиляция тестовой программы
$ export LD_LIBRARY_PATH=.; ./test_complex         # запуск тестовой программы
```

- Есть ряд стандартных путей (), где система ищет динамически загружаемые библиотеки. Текущая директория по умолчанию не является таким местом, поэтому мы явно указываем, что библиотеки могут лежать также и в текущей директории. Переменная `LD_LIBRARY_PATH` содержит дополнительный список путей (разделённых двоеточием), где система будет искать библиотеки.
- Расширение `.so` соответствует `shared objects`, то есть библиотека объектов общего доступа.

Подключение библиотеки в виде статически загружаемой библиотеки

Для компилятора Gnu C команды создания статически подключаемой библиотеки и подключения её к проекту выглядят следующим образом

```
$ gcc -c complex.c -o libcomplex.a                # создание статически подключаемой библиотеки
$ gcc test_complex.c -L. -lcomplex -o test_complex # компиляция тестовой программы
$ ./test_complex                                  # запуск тестовой программы
```

Задачи

- Напишите расширенную версию библиотеки функций для работы с комплексными числами. Реализуйте операции сложения, вычитания, умножения и деления, а также функции, вычисляющие аргумент и модуль комплексного числа. Реализуйте функции комплексного аргумента sinc, cosc, expc, logc, asinc , acosc. Чему равен арккосинус двух (acosc(2) = ?) ? Напишите программу, тестирующую вашу библиотеку.
- Напишите библиотеку функций по теме "аналитическая геометрия"
  - Решать задачи на плоскости
    - вычислять расстояние для пар (точка, точка), (точка, прямая)
    - находить площадь многоугольника
    - проверять, пересекаются ли две прямые и находить точку их пересечения
    - проверять, пересекаются ли два отрезка и находить точку их пересечения
  - Решать задачи в пространстве
    - вычислять расстояние для пар (точка, точка), (точка, прямая), (прямая, прямая), (точка, плоскость)
    - находить точку пересечения трёх попарно непараллельных плоскостей
    - находить объем тетраэдра

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Библиотека_complex&oldid=110241

Эта страница в последний раз была отредактирована 13 апреля 2015 в 06:10.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Сортировка на основе qsort

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Пример использования функции `qsort` из `stdlib`

Вход: В первой строчке дано N, а затем следует N строчек вида ИМЯ ТЕЛЕФОН

Выход: Выведите телефонный справочник в алфавитном порядке и в порядке возрастания телефонных номеров.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct person_t {
   char * name;
   int phone;
} person_t;

int namecmp(const void* a, const void* b);
int phonecmp(const void* a, const void* b);

int main(void)
{
   char name[1000];
   person_t *base;
   int n, i, phone;
   scanf("%d", &n);
   base = malloc(n * sizeof(person_t));

   /* Считываем телефонную базу */
   for(i = 0 ; i < n ; i++)
   {
       scanf("%s%d", name, &phone);
       /* Выделяем память под массив base[i].name,
                    учитывая символ конца строки */
       base[i].name = malloc((strlen(name)+1) * sizeof(char));
       strcpy(base[i].name, name);
       base[i].phone = phone;
   }

   qsort(base, n, sizeof(person_t), namecmp);
   /* Напечатаем в алфавитном порядке */
   for(i = 0 ; i < n ; i++)
   {
       printf("%20s %d\n", base[i].name, base[i].phone);
   }


   qsort(base, n, sizeof(person_t), phonecmp);
   /* Напечатаем в порядке возрастания номеров*/
   for(i = 0 ; i < n ; i++)
   {
       printf("%20s %d\n", base[i].name, base[i].phone);
   }

   /* Освобождаем выделенную память */


   for(i = 0 ; i < n ; i++)
   {
       free(base[i].name);
   }
   free(base);
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio

```
    return 0;

}

int namecmp(const void* a, const void* b)
{
    person_t *pa = a;
    person_t *pb = b;
    return strcmp(pa->name, pb->name);
}
int phonecmp(const void* a, const void* b)
{
    person_t *pa = a;
    person_t *pb = b;
    return pa->phone - pb->phone;
}
```

24. Декодирование звукозаписи в формате ADX

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Сортировка_на_основе_qsort&oldid=93330

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:48.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Калькулятор выражений в обратной польской нотации

Материал из Викиучебника — открытых книг для открытого мира

Содержание

Что такое обратная польская нотация?
- Пример работы программы
- Задания
- Замечания
- Реализация калькулятора с явным определением операций со стеком
- Выделение стека в отдельную структуру
- Задания

Что такое обратная польская нотация?

Рассмотрим запись арифметических выражений, в которых сначала следуют два операнда арифметической операции, а затем знак операции. Например:

Обратная польская нотация	Обычная нотация
2 3 +	2 + 3
2 3 * 4 5 * +	(2 * 3) + (4 * 5)
2 3 4 5 6 * + - /	2 / (3 - (4 + (5 * 6)))

Это нотация записи выражений называется обратной польской нотацией (записью) (Reverse Polish Notation, RPN). В теории языков программирования эта нотация называется *постфиксной нотацией*. Обычная нотация называется алгебраической или *инфиксной нотацией* («ин» от англ. *inside*, то есть между операндами). Есть также префиксная нотация, активно используемая в языке Си (сначала имя функции, а затем её аргументы), а также в языке LISP.

Заметьте, что скобки в обратной польской нотации не нужны. В частности, если во втором примере мы опустим скобки, выражение по-прежнему будет интерпретироваться однозначно.

Транслятор RPN-выражений основан на стэке. Каждое следующее число помещается в стэк. Если встречается знак операции (обозначим его *), то два числа из стека извлекаются (a =

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в

pop(), b = pop()), для них вычисляется значение соответствующей бинарной арифметической операции, и результат помещается в стек (push(a * b)).

формате ADX

Задача: Напишите программу-калькулятор арифметических выражений записанных в обратной польской нотации.

Ниже приведена простая реализация такого калькулятора на языке Си. Роль стека играет обычный массив (stack[1000]). Переменная sp (stack pointer) равна количеству элементов в стеке. Число (sp - 1) равно индексу ячейки, являющейся вершиной стека.

```c
#include <stdio.h>
int main()
{
    int stack[1000];
    // sp = индекс ячейки, куда будет push-иться очередное число
    int sp =0;      // (sp-1) = индекс ячейки, являющейся вершиной стека
    while ( !feof(stdin) ) {
        int c = getchar();
        int x;
        switch (c) {
            case  ' ':
            case '\n':
                break;
            case '=':
                printf("Result = %d\n", stack[sp - 1]);  sp--;
                break;
            case '+':
               stack[sp-2] = stack[sp-2] + stack[sp-1];  sp--;
               break;
            case '-':
               stack[sp-2] = stack[sp-2] - stack[sp-1];  sp--;
               break;
            case '*':
               stack[sp-2] = stack[sp-1] * stack[sp-2];  sp--;
               break;
            case '/':
              stack[sp-2] = stack[sp-2] / stack[sp-1];   sp--;
               break;
            default:
                ungetc (c, stdin); // вернуть символ обратно в поток
                if (scanf("%d", &x) != 1) {
                    fprintf(stderr, "Can't read integer\n");
                    return -1;
                } else {
                    stack[sp] = x;                       sp++;
                }
        }
    }
    printf("Result = %d\n",stack[sp-1]);
    return 0;
}
```

Пример работы программы

```
> ./stack
1 2 3 4 + * + =
Result = 15
1 2 + 3 4 + * =
Result = 21
```

Задания

1. Введите входные следующие входные данные `1 2 3 * * * * = = = =`. К чему это привело? Добавьте к приведенной программе «защиту от дурака».
2. Введите входные данные состоящие из 10000 единиц и 9999 знаков +. Сможет ли программа отработать на этих входных данных?

Замечания

1. У данной программы есть целый ряд недостатков:

    - Программа не будет работать, если количество элементов в стеке превысит 1000
    - Программа не замечает некорректные входы и отрабатывает их с непредсказуемыми последствиями.
    - В программе явно не отображено, что присутствует структура данных «стек». Это затрудняет понимание логики работы этой программы и усложняет сторонним разработчикам доработку данной программы.

Реализация калькулятора с явным определением операций со стеком

Введем операции работы со стеком в программу. Это повысит читаемость кода и облегчит понимание заложенной в программу логики.

```
#include <stdio.h>
#include <malloc.h>

int sp = 0;
int stack[1000];
int pop(void) {
    if (sp > 0) {
         return stack[--sp];
    } else {
         fprintf(stderr, "Невозможно выполнить POP для пустого стека.\n");
         return 0;
    }
};
void push(int a) {
    stack[sp++] = a;
};
int empty() {
    return (sp == 0);
}

int main() {
    int i;
    while (!feof(stdin)) {
        int c = getchar();
        int x;
        switch (c) {
            case '\n':
            case ' ' : break;
            case '=' : printf("Result = %d\n", pop()); break;
            case 27  : goto RESULT;
            case '+' : push(pop() + pop()); break;
            case '-' : push(-pop() + pop()); break;
            case '*' : push(pop() * pop()); break;
            default:
                ungetc(c, stdin);
                if (scanf("%d", &x) != 1) {
                    fprintf(stderr, "Can't read integer\n");
                    return -1;
                } else {
                    push(x);
```

```
                }
                break;
            }
        }
    RESULT:
        i = 0;
        while ( !empty() ){
            printf("Stack[%d] = %d\n", i,  pop());
            i++;
        }
        return 0;
    }
```

В данной программе в некоторой степени реализована «защита от дурака», а именно, если вводится выражение, в котором число операций превосходит число помещенных в стек элементов (например `1 2 + *`), то программа не допустит уменьшения переменной `sp` до отрицательных значений, а выдаст предупреждение «Невозможно выполнить POP для пустого стека»..

Выделение стека в отдельную структуру

Кроме защиты от дурака-пользователя, необходима еще защита от дурака-программиста, который возьмет ваш код, решит его использовать и дорабатывать. Точнее, нужно просто соблюдать некоторые правила, которые не позволили бы программисту, который включил ваш код в свой проект, получить ошибки, связанные с пересечением имён, испольуемых им в своих файлах и вами, в файле `stack.c`.

В первую очередь, в принципе не рекомендуется объявлять глобальные переменные, особенно такие, которые не являются всеобщим достоянием, а относятся к внутренним делам отдельного модуля (например, нашей программы, для работы со стеком).

Кроме того, рекомендуется имена функций, присутствующие в библиотеке, снабжать префиксом. Например, имена функций, связанных со стеком, логично снабдить префиксом `stack`.

В итоге получаем следующую, «более правильную» реализацию стека:

```
    #include <malloc.h>
    /* Структура с указателем на массив int и служебной информацией
     */
    typedef struct stack {
        int *data;
        int last;
        int data_size;
    } stack_t;
    /* Конструктор стека.
     * Принимает начальный размер стека и возвращает указатель на новый стек
     */
    stack_t* stack_new(int initial_size) {
        stack_t *stack = (stack_t*) malloc(sizeof(stack_t));
        stack->data_size = initial_size;
        if(stack->data_size <= 0) stack->data_size = 100;
        stack->last = 0;
        stack->data = (int*) malloc(sizeof(int)* stack->data_size);
        return stack;
    }
    /* Деструктор стека.
     * Принимает на вход указатель на стек и освобождает занятую им память.
     */
    void stack_delete(stack_t *stack) {
        free(stack->data);
        free(stack);
    }
    /* Операция push
     * Принимает указатель на стек и добавляемый элемент.
     * При необходимости увеличивает количество памяти для  массива элементов.
     */
    void stack_push(stack_t *stack, int a) {
```

```
        if (stack->last == stack->data_size) {
            stack->data_size = (stack->data_size * 3 + 1) / 2;
            stack->data = (int*)realloc(stack->data,  stack->data_size * sizeof(int));
        }
        stack->data[stack->last++] = a;
    }
    /* Операция pop
     * Принимает указатель на стек и адрес значения верхнего элемента.
     * Возвращает 1 в случае, если стек был непуст, и 0 -- в противном случае.
     */
    int stack_pop(stack_t *stack, int *a) {
        if (stack->last > 0) {
            *a = stack->data[--stack->last];
            return 1;
        } else {
            fprintf(stderr, "Попытка извлечь элемент из пустого стека!\n");
            return 0;
        }
    }
```

Задания

1. Реализуйте RPN-калькулятор, используя эту реализацию стека. Выделите вычисление RPN-выражения в отдельную функцию.
2. Постройте систему тестов и проверьте, что ваш калькулятор успешно проходит все тесты и «защищён от дурака» (как дурака-пользователя программы, так и дурака-программиста, использующего ваш стек и калькулятор).

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Калькулятор_выражений_в_обратной_польской_нотации&oldid=157684

Эта страница в последний раз была отредактирована 11 октября 2019 в 21:52.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Калькулятор выражений в обратной польской нотации на Bison

Материал из Викиучебника — открытых книг для открытого мира

Содержание

Калькулятор на Bison
- Файл с правилами bison
- Файл на языке Си с функциями main и yylex
- Команды компиляции
- Пример использования калькулятора:

Калькулятор на Bison

GNU Bison

Файл с правилами `bison`

```
/*
  file:  rpn.yy
  title: Reverse polish notation calculator rules.
*/

%{
     #define YYSTYPE double
     #include <math.h>
     #include <stdio.h>
%}

%token NUM

%% /* Grammar rules and actions follow */


     input:    /* empty */
             | input line
     ;

     line:     '\n'
             | exp '\n'  { printf ("\t%.10g\n", $1); }
     ;

     exp:      NUM             { $$ = $1;         }
             | exp exp '+'     { $$ = $1 + $2;    }
             | exp exp '-'     { $$ = $1 - $2;    }
             | exp exp '*'     { $$ = $1 * $2;    }
             | exp exp '/'     { $$ = $1 / $2;    }
           /* Exponentiation */
             | exp exp '^'     { $$ = pow ($1, $2); }
           /* Unary minus    */
             | exp 'n'         { $$ = -$1;        }
     ;
%%
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio

Примечание: Пример реально удалось запустить... НО... при выполнении этого примера в VM-Linux под SuSe-11.1 относительно оригинала статьи пришлось сделать следующие изменения:

24. Декодирование звукозаписи в формате ADX

1. В данный файл описания грамматики(сразу после "#include <stdio.h>" и до "%}") пришлось добавить прототипы ф-ий "yylex", "yyerror" (определенных в «main»):

```
        int yylex (void);
        int yyerror (const char *s);
```

2. Вместо команды «cpp» - трансляцию-сборку всего примера пришлось выполнять при помощи команды «c++».

Файл на языке Си с функциями `main` и `yylex`

```
/*
  file:  rpn.cc
  title: Tokenizer functions
         yylex   -- функция разбиения входного потока на токены.
         yyerror -- функция обработки ошибок
         main    -- главная функция
*/
#include <stdio.h>
#include <ctype.h>
#include "rpn.tab.hh"
int yyparse(void);

int yyerror (const char *s)  /* Called by yyparse on error */
{
    printf ("%s\n", s);
}

/* Lexical analyzer returns a double floating point
    number on the stack and the token NUM, or the ASCII
    character read if not a number.  Skips all blanks
    and tabs, returns 0 for EOF. */
int yylex (void)
{
    int c;

    /* skip white space  */
    while ((c = getchar ()) == ' ' || c == '\t')
      ;
    /* process numbers   */
    if (c == '.' || isdigit (c))
    {
        ungetc (c, stdin);
        scanf ("%lf", &yylval);
        return NUM;
     }
     /* return end-of-file  */
     if (c == EOF)
        return 0;
     /* return single chars */
     return c;
}

int main (void)
{
    return yyparse ();
}
```

Команды компиляции

```
$ bison -d rpn.yy      # -> produce files rpn.tab.cc  and rpn.tab.hh
$ gcc rpn.tab.cc rpn.cc -lm -o rpn #  produce executable file rpn
```

Пример использования калькулятора:

```
$
./rpn
1 2 +
        3
1 2 3 * + 100 +
        107
<Ctrl+D>
$
```

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Калькулятор_выражений_в_обратной_польской_нотации_на_Bison&oldid=93334

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:50.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Простая грамматика

Материал из Викиучебника — открытых книг для открытого мира
< Язык Си в примерах

Грамматики (подмножества множества слов в некотором алфавите) можно описывать с помощью правил. Каждое правило состоит из левой и правой части, между которыми стоит стрелка (`->`).

Данную стрелку можно интерпретировать как «представляет собой» или «состоит из». Например, правило

```
A -> B C D
```

можно прочитать как

> «Слово типа A **представляет собой** слияние слова типа B, слова типа C и слова типа D.»

Но в теории формальных грамматик принята другая терминология:

> «Из символа A **можно вывести** последовательность символов B, C и D.»

Рассмотрим три правила, в которых присутствует только один символ S (один нетерминальный символ), и три терминальных (то есть нераскладываемых, таких символов, из которых ничего нельзя вывести кроме их самих) символов `'0', '1', '2', '3'`:

```
S -> '0'
S -> '1' S
S -> '2' S S
S -> '3' S S S
```

Правило `S -> '3' S S S` , к примеру, означает, что если *a*, *b* и *c* являются корректными словами (словами, выводимыми из символа S), то и слово 3*abc* тоже является корректным.

- Примеры корректных слов: 0, 10, 110, 200, 2100, 2010, 1111111110.
- Примеры некорректных слов: 01, 00, 300, 3333, 11120.

Приведённая ниже программа на Си определяет корректность введённого слова.

```
#include <stdio.h>
#include <limits.h>
int Read()
{
    int head, i;
    scanf("%d", &head);            // Читаем первое число.
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

```
    if(head < 0 || head > 3) {    // Возвращаем код ошибки, если
оно не из
        return 1;                 //     множества {0, 1, 2, 3}
    }
    for(i = 0; i < head ; i++) {
       if(Read()) {
           return 1;
       }
    }
    return 0;
}
int main()
{
  if(!Read()) {
     int n;
     if(scanf("%d", &n) != 0)
        printf("incorect\n");
     else
        printf("correct\n");
  } else {
     printf("incorrect\n");
  }
  return 0;
}
```

Здесь представлен классический рекурсивный способ лексографического разбора.

Программный код можно максимально приблизить к самим правилам:

```
ReadS() {
    if( scanf("%d", &n) != 1 ) return 0;
    switch(c) {
      '0': return 1;
      '1': return ReadS();
      '2': return (ReadS() && ReadS() );
      '3': return (ReadS() && ReadS() && ReadS());
       default: return 0;
    }
}
```

Задание

Задача 1. Напишите программу, определяющую корректность слова, не используя идеи рекурсии и стека. Для этого обявите переменную `n`, равную количеству объектов `S`, которые осталось считать. Изначально `n = 1`.
Задача 2. Напишите программу, которая определяет, является ли введённое слово выводимым из символа S.

```
S ->  A | B
A -> '(' B* ')'
B -> '[' A* ']'
```

Вертикальная черта «|» означает соединительный союз «или».

Здесь используется специальный символ «*». Это указатель количества. Запись `B*` означает любое количество слов типа `B` (слов, выводимых из символа `B`), либо пустое слово. Другими словами, символ «*» означает 0, 1, 2,... ∞ раз. Есть также специальные символы + и ?:

- ***** -- 0,1, ..., ∞ раз.
- **+** -- 1,2, ..., ∞ раз (то есть, любое число раз, но по крайней мере один раз).
- **?** -- 0 или 1 раз (то есть либо есть, либо нет).

Примеры слов, выводимых из A:

```
(), ([]), ([][()()]), ([][][][])
```

Примеры слов, выводимых из B:

```
[], [()], [([])([])()], [([][][][])]
```

Подсказка: определите две функции ReadA и ReadB которые рекурсивно вызывают друг друга.

Код примерно должен быть таким:

```
ReadChar(char x) {
   int c;
   if((c=getchar()) != x) {
       ungetc(c);
       return 0;
   } else {
       return 1;
   }
}
ReadA() {
    int c;
    if ( !ReadChar( '(' ) ) return 0;  // читаем открывающую скобку
    while(ReadB()) {};                 // читаем B*
    if ( !ReadChar( ')' ) ) return 0;  // читаем закрывающую скобку
    return 1;
}
ReadB() {
    ... // по аналогии с ReadA
}
ReadS() {
    ReadA || ReadB;
    if(getchar() != EOF)
        printf ("Incorrect\n");
    else
        printf ("Correct\n");
}
```

Разбор языков (parsing), которые задаются простыми рекурсивными грамматиками, реализуют с помощью рекурсивных функций, которые возвращают 1 (успешно считано) или 0 (не считано).

Задача 4. Прочитайте учебный текст про формальные грамматики. Напишите грамматики следующих языков:

- язык записи действительных чисел в языке Си.
- язык записи email адресов.
- язык записи арифметических выражений над целыми числами со скобками и операциями + * - / .

Задача 5. Просмотрите формальное описание грамматики языка Си. Является ли условие выводимости из символа program необходимым для компиляции программы? Является ли это условие достаточным? Ответ обоснуйте.

Задача 6. Изучите задачу вычисления сопротивления по описанию параллельно-последовательной электрической схемы из сопротивлений — Язык Си в примерах/ Задача «Расчёт сопротивления схемы»

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Простая_грамматика&oldid=93340

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:54.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Задача «Расчёт сопротивления схемы»

Материал из Викиучебника — открытых книг для открытого мира

На данной задаче можно отработать технику разбора простых грамматик.

Параллельно-последовательная схема сопротивлений (ПП-схема) это

- параллельно соединённые ПП-схемы (записывается как "[s1 s2 .. sk]", k = 1, 2, ...),

ИЛИ

- последовательно соединённые ПП-схемы (записывается как "(s1 s2 ... sm)", m = 1, 2, ..),

ИЛИ

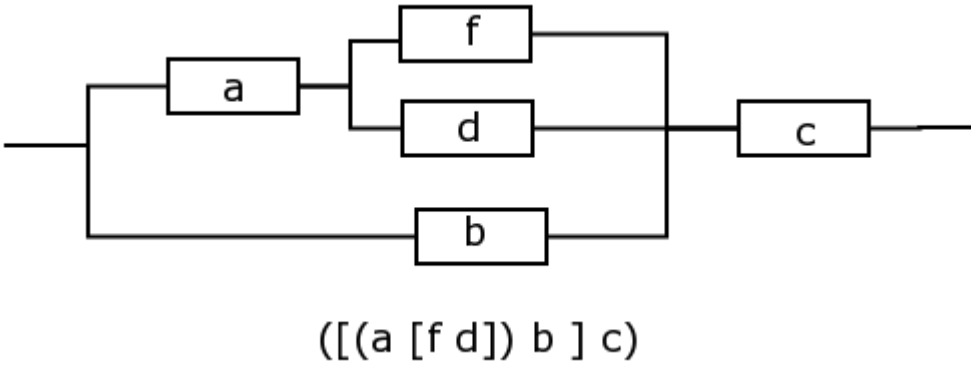
- один резистор (обозначается действительным числом "R").

На рис.1 приведён пример ПП-схемы и её описания в виде скобочной структуры.

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio

24. Декодирование звукозаписи в формате ADX



([(a [f d]) b] c)

Рисунок 1. Пример ПП-схемы и её описания в виде скобочной структуры.

Запишем грамматику описания ПП-схем в нотации Бэкуса — Наура, расширенной регулярными выражениями:

```
S ::= A | B | number.
A ::= '(' S+ ')' .
B ::= '[' S+ ']' .
number ::= digit+ ('.' digit+ )? .
digit ::= '0' | '1' | '2' | '3' | '4' |  '5' | '6' | '7' | '8' | '9' .
```

- Вход. Строчка с описанием схемы.

Значения сопротивлений — действительные числа из отрезка [0, 100000]. Размер входа не превосходит 200000 байт.

- Выход. Значение сопротивления схемы (с точностью до 0.1%).

```
Вход#1                      Выход#1
[2 2 2]                     0.6666667
```

```
Вход#2                      Выход#2
([1 1] 0.5 [3.2 (1 1)])     2.230769
```

```
Вход#3                      Выход#3
(3 0 1)                     4
```

```
    #include <stdio.h>
    int read_S(double*);
    int read_A(double*);
    int read_B(double*);
    void read_space() {
        int c ;
        do { c = getchar(); } while ( c==' ' || c=='\n' || c=='\t' );
        if( c != EOF ) ungetc(c, stdin);
    }
    int read_number(double *r) {
        return ( scanf("%lf", r) == 1 );
    }
    int read_A(double *r) {
        int c;
        if( (c=getchar()) == '(') {
            double q;
            *r = 0;
            if( ! read_S(r) ) return 0;
```

```
            while( read_S(&q) ) {
                *r += q;
            }
            if( (c=getchar()) == ')') {
                return 1;
            } else {
                return 0;
            }
        } else {
            ungetc(c, stdin);
            return 0;
        }
    }
    int read_B(double *r) {
        int c;
        if( (c=getchar()) == '[' ) {
            double q;
            *r = 0;
            if( ! read_S(r) ) return 0;
            *r = 1 / *r;
            while( read_S(&q) ) {
                if(q != 0) *r += 1 / q;
            }
            *r = 1 / *r;
            if( (c=getchar()) == ']' ) {
                return 1;
            } else {
                return 0;
            }
        } else {
            ungetc(c, stdin);
            return 0;
        }
    }
    int read_S(double *r) {
        read_space();
        return read_A(r) || read_B(r) || read_number(r);
    }
    int main() {
        double r;
        read_S(&r);
        printf("%lf\n", r);
        return 0;
    }
```

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Задача_«Расчёт_сопротивления_схемы»&oldid=120356

Эта страница в последний раз была отредактирована 13 августа 2015 в 13:42.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Простая реализация конечного автомата

Материал из Викиучебника — открытых книг для открытого мира

Ниже приведена реализация конечного автомата, который прибавляет 1 к числу, поданному на вход в виде двоичной записи начиная с младшего разряда.

```
#include <stdio.h>
int c;

int
main()
{
    goto s1;

s2:  c = getchar();

    switch(c)
        {
         case EOF:
            exit(0);
         default :
            putchar(c);
            goto s2;
        }
s1:  c = getchar();
    switch (c)
        {
        case EOF:
            exit(0);
        case '1':
            putchar('0');
            goto s1;
        case '0':
            putchar('1');
            goto s2;
        }

}
```

```
#include <stdio.h>

int char_to_id (int c) {
    switch (c) {
        case '0': return 0;
        case '1': return 1;
        case EOF: return 2;
        default: return 2;
    }
}

typedef struct table_item_s {
    int state;
    int out_char;

} table_item_t;

#define END_STATE 2
table_item_t
T[2][3] = {
    { {1, '1'}, {0, '0'} , {END_STATE, '\n'}},
    { {1, '0'}, {1, '1'} , {END_STATE, '\n'}}
};


 int main() {
    int c, c_id;
    int state = 0;
    while(!feof(stdin)) {
        c = getchar();
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio

```
            c_id  = char_to_id(c);
            putchar(T[state][c_id].out_char);
            state = T[state][c_id].state;
            if(state == END_STATE)
                return 0;
        }
    }
```

24. Декодирование звукозаписи в формате ADX

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Простая_реализация_конечного_автомата&oldid=93336

Эта страница в последний раз была отредактирована 1 июня 2014 в 17:51.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Использование аргументов командной строки

Материал из Викиучебника — открытых книг для открытого мира

Содержание

Аргументы командной строки
Пример 1
Подробнее об аргументах командной строки
Пример 2
Задачи для самостоятельного решения
См. также

Аргументы командной строки

Программы могут принимать аргументы. Ниже приведен пример программы `argv`, которая печатает список аргументов, которые были переданы ей в командной строке. Пример ее работы:

```
bash$ ./argv alpha beta gamma last
Argument 0: ./argv
Argument 1: alpha
Argument 2: beta
Argument 3: gamma
Argument 4: last
```

Пример 1

```
 /*
    Build me with
      gcc -o argv argv.c
 */
 #include <stdio.h>
 int main (int argc, char * argv[]) {
         int i;
         for( i = 0 ; i < argc; i++) {
                 printf("Argument %d: %s\n", i, argv[i]);
         }
         if(argc == 1) {
                 printf("Command line has no additional
arguments\n");
         }
         return 0;
 }
```

Например стандартная программа 'ls' в Linux в качестве аргумента принимает имена директорий, содержимое которых нужно вывести в стандартный поток вывода. Команда

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в

```
bash$ ls /home/user
```

формате ADX

выведет содержимое директории '/home/user'.

Команда 'ls' имеет множество опций, которые можно передавать в командной строке. О них вы можете узнать, запустив команду 'ls' с опцией '-h':

```
bash$ ls -h
```

Вообще, программа может получать данные из нескольких источников. Наиболее важные источники это:

- аргументы командной строки;
- стандартный поток ввода;
- файлы;
- средства взаимодействия процессов (signals, sockets, messages, ...).

Научимся использовать первый из них.

Подробнее об аргументах командной строки

Основной функцией в программе на языке Си является функция `main()`. Мы уже познакомились с простейшим способом объявления этой функции:

```c
/*
* Простейшая форма main()
*/
int main() {

    // ...

    return 0;
}
```

Для того, чтобы ваша программа могла анализировать параметры, передаваемые ей через командную строку, функцию main() следует описать несколько сложнее:

```c
/*
* main() с обработкой параметров
*/
int main(int argc, char **argv) {

    // ...

    return 0;
}
```

При такой записи, при вызове вашей программы, параметр `argc` будет содержать количество параметров командной строки **плюс один**. Действительные параметры нумеруются с единицы по `argc-1` включительно (т.е. всего `argc-1` параметров), параметр под номером ноль является служебным — он содержит полный путь и имя исполняемого файла вашей программы.

Для доступа к параметру под номером `i` следует писать `argv[i]`. Выражение `argv[i]` — это строка, передаваемая в соответствии с соглашениями языка Си о строках. С ней можно проделывать все операции, как с обычной строкой (её даже можно модифицировать, но это

считается дурным тоном!).

Вот еще один пример программы, работающей со своей командной строкой:

Пример 2

```c
/*
* Пример программы, работающей со своей командной строкой
*/
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
    if(argc <= 1) {
        printf("syntax : \n\t%s --help or\n\t./p --add [a] [b]\n", argv[0]);
        return 0;
    }
    if(!strcmp(argv[1], "--help")) {
        printf("You requested help message.\n");
    } else if(!strcmp(argv[1], "--add")) {
        if(argc <= 3) {
            printf("'--add' operation requires two parameters.\n");
        } else {
            int a, b;
            if(sscanf(argv[2], "%d", &a) != 1 || sscanf(argv[3], "%d", &b) != 1) {
                printf("'--add' operation requires two integer parameters.\n");
            } else {
                printf("%d + %d = %d\n", a, b, a+b);
            }
        }
    } else {
        printf("Unknown parameter: '%s'. Type %s --help for help.\n", argv[1], argv[0]);
    }
    return 0;
}
```

Задачи для самостоятельного решения

- Напишите программу, которая интерпретирует свои аргументы как целые числа и выводит на стандартный поток вывода их сумму.
- Напишите программу, которая принимает аргумент -- имя файла, и выводит содержимое этого файла.
    - в текстовом виде, обозначая не ASCII символы знаками вопроса (если дана опция =-text=);
    - в шестнадцатиричном виде (если дана опция =-hex=);
    - как есть (по умолчанию).

См. также

- Системы счисления

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Использование_аргументов_командной_строки&oldid=146513

Эта страница в последний раз была отредактирована 23 мая 2018 в 08:28.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Чтение и печать целого числа без использования stdio

Используя функции scanf и printf можно считывать форматированный ввод и печатать форматированный вывод. Их возможности очень велики.

`scanf` и `printf` не есть команды языка C. Они являются функциями, определенными в стандартной библиотеке ANSI C.

Возникает вопрос: Как они описаны в этой библиотеке?

Ответ: Есть более низкоуровневые функции для считывания и печати данных, но они платформозависимые (зависят от архитектуры компьютера и операционной системы).

В OS Linux присутствуют системные вызовы `read` и `write`, которые читают (пишут) указанное количество байт из (в) указанный поток.

Аргументы у этих системных вызовов одинаковые, их три штуки:

- идентификатор потока ввода/вывода (дескриптор потока)
- адрес, куда нужно помещать или откуда нужно брать байты
- максимальное количество байт, которое мы хотим считать (напечатать)

Стандартный поток ввода имеет идентификатор (дескриптор потока) равный 2. Стандартный поток вывода имеет идентификатор 1.

Возвращаемое значение равно количеству успешно записанных (считанных) байт.

```
#include <sys/syscall.h>
#define N 100

char buffer[N];

void print_int (int);
int  scan_int  (void);


main()
{
        int a = scan_int();
        int b = scan_int();

        print_int(a+b);
       return 0;
}
```

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio

```
void print_int(int a)
{
        char minus = '-';
        if(a < 0) {
                write(1, &minus,  1);
                a = -a;
        }
        int i = N;
        buffer[--i] = '\n';
        while(a)
        {
                buffer[--i] = '0' +  (a % 10);
                a /= 10;
        }
        write(1, &(buffer[i]), N - i);

}

int scan_int()
{
        int i = 0, res = 0, sign = 1;
        int n = read(2, buffer, N);
        if (buffer[i] == '-') {
            sign = -1;
            i++;
        }
        for(; i < n ; i++)
        {
                if( buffer[i] >= '0' && buffer[i] < '9' )
                {
                        res *= 10;
                        res += buffer[i] - '0';
               } else {
                        break;
                }
        }
        return res*sign;

}
```

24. Декодирование звукозаписи в формате ADX

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Чтение_и_печать_целого_числа_без_использования_stdio&oldid=147276

Эта страница в последний раз была отредактирована 20 июля 2018 в 14:05.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.

Язык Си в примерах/Декодирование звукозаписи в формате ADX

Материал из Викиучебника — открытых книг для открытого мира

ADX — формат сжатия звукозаписи с потерями, разработанный для игровых приставок Sega Dreamcast, Sony PlayStation и GameCube.

Заголовок файла

Формат диска ADX определён в порядке байт от старшего к младшему. Опознанные разделы главного заголовка

Язык Си в примерах

1. Компиляция программ
2. Простейшая программа «Hello World»
3. Учимся складывать
4. Максимум
5. Таблица умножения
6. ASCII-коды символов
7. Верхний регистр
8. Скобочки
9. Факториал
10. Степень числа
11. Треугольник Паскаля
12. Корень уравнения
13. Система счисления
14. Сортировка
15. Библиотека complex
16. Сортировка на основе qsort
17. RPN-калькулятор
18. RPN-калькулятор на Bison
19. Простая грамматика
20. Задача «Расчёт сопротивления схемы»
21. Простая реализация конечного автомата
22. Использование аргументов командной строки
23. Чтение и печать без использования stdio
24. Декодирование звукозаписи в формате ADX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x0	0x80	0x00	Смещение данных об авторских правах		Тип кодирования	Размер блока	Глубина образца в битах	Счётчик каналов	Частота выдачи образцов				Общее количество образцов			
0x10	Частота среза верхних частот		Версия	Флаги	Неизвестно				Зацикливание включено (v3)				Указатель на образец в начале цикла (v3)			
0x20	Указатель на байт в начале цикла (v3)				Зацикливание включено (v4) Указатель на образец в конце цикла (v3)				Указатель на образец в начале цикла (v4) Указатель на байт в конце цикла (v3)				Указатель на байт в начале цикла (v4)			
0x30	Указатель на образец в конце цикла (v4)				Указатель на байт в конце цикла (v4)				Ноль или больше байт пустого пространства							
???	[Смещение данных об авторских правах (CopyrightOffset) - 2] строка ASCII (unterminated): "(c)CRI"															
...	[Смещение данных об авторских правах (CopyrightOffset) + 4] Здесь начинаются данные звукозаписи															

Поля, помеченные как «неизвестно» содержат либо неизвестные данные, либо просто зарезервированы (например, заполнены нулевыми байтами). Поля, помеченные «v3» или «v4», но не обеими версиями, равнозначны «неизвестным» в той версии, к которой они не относятся. Также следует обратить внимание на то, что этот заголовок может быть укорочен до 20 байт (0x14), что при задании авторских прав неявно удаляет поддержку цикла, так как эти поля не представлены («затираются»).

Поле «тип кодирования» должно содержать одно из значений:

- **0x03** для стандартного ADX
- **0x04** для ADX с экспоненциальной шкалой (масштабом)
- **0x10** или **0x11** для AHX

Поле «версия» должно содержать одно из значений:

- **0x02** для одной из 'версии 3' с различными чётко указанными декодировщиками
- **0x03** для ADX 'версии 3'
- **0x04** для ADX 'версии 4'
- **0x05** для версии ADX 4 без внутренней поддержки цикла

При декодировании звукозаписи в формате AHX , считывание значения в поле «версия» не представлено и может быть безопасно пропущено.

Формат образца

Звукозапись, кодированная в ADX разбита на несколько последовательностей 'блоков', каждая из которых содержит данные только для одного канала. Блоки затем раскладываются в 'кадры', каждый из которых состоит из набора по одному блоку из каждого канала в порядке возрастания. Например, в стерео (2 канала) поток будет состоять из: кадра 1: блок левого канала, блок правого канала; кадра 2: левый канал, правый канал; и т.д. Обычный размер блока: 18 бит, содержащих образцы по 4 бита или других технически возможных размеров. Образец блока выглядит так:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Масштаб		32 4-битных образца															

Масштаб — это 16-ти битовая беззнаковая целочисленная переменная [integer] (с байтами идущими от старшего к младшему, как и в заголовке), которая по существу значением является усиления всех образцов в этом блоке. Каждый образец должен быть декодирован в порядке потока бит, так что в начале идёт сам бит знак переменной. Например, когда размер образца составляет 4 бита:

7	6	5	4	3	2	1	0
Первый образец				Второй образец			

Сами по себе образцы не обращены, так что нет необходимости дополнительно работать с ними после извлечения. Каждый образец имеет знак, так что для этого примера значение может быть в диапазоне между -8 и +7 (который может расширяться масштабом во время декодирования). С другой стороны, благодаря заголовку, допустима любая глубина между 1 и 255. Маловероятно, что когда-либо появятся однобитовые образцы, так как они могут представлять только значения {0, 1}, {-1, 0} или {-1, 1}, каждое из которых малопригодно для кодирования музыки — если такие всё-таки появятся, неизвестно какой из трёх вариантов набора значений верен для интерпретации.

Декодирование ADX

Этот раздел посвящён декодированию ADX версии 3 или 4, в котором указатель «тип кодирования» установлен в «стандартный ADX» (**0x03**). Кодировщик можно построить, «перевернув» алгоритм, указанный в дальнейшем коде. Все примеры кода написаны с использованием C99.

Перед тем, как кодировать/декодировать стандартный ADX, необходимо вычислить коэффициенты предсказания. В основном это лучше делать на начальном этапе:

```
 #define M_PI acos(-1.0)
 double a, b, c;
 a = sqrt(2.0) - cos(2.0 * M_PI * ((double)adx_header->highpass_frequency / adx_header->sample_rate));
 b = sqrt(2.0) - 1.0;
 c = (a - sqrt((a + b) * (a - b))) / b; //(a+b)*(a-b) = a*a-b*b, в любом случае, более простая формула теряет в точности числа с плавающей запятой

 // удвоение coefficient[2];
 coefficient[0] = c * 2.0;
 coefficient[1] = -(c * c);
```

Этот код рассчитывает коэффициенты предсказания для предсказания текущего образца из 2 предыдущих образцов.

Когда мы знаем коэффициенты декодирования, мы можем начинать декодировать поток:

```
 static int32_t*      past_samples; // В начале обнуляются ранее декодированные образцы каждого канала (size = 2*channel_count)
 static uint_fast32_t sample_index = 0; // sample_index это указатель последовательности образцов, которую в дальнейшем необходимо декодировать
 static ADX_header*   adx_header;

 // buffer (буфер) это область памяти, в которую мы будем помещать декодированные образцы
 // samples_needed устанавливает, как много 'последовательностей' образцов (один образец из каждого канала), чтобы заполнить буфер (buffer)
 // looping_enabled это логический флаг для управления использованием встроенного цикла
 // Возвращает количество 'последовательностей' образцов в буфер (buffer), которое не может его заполнить (EOS)
 unsigned decode_adx_standard( int16_t* buffer, unsigned samples_needed, bool looping_enabled )
 {
  unsigned const samples_per_block = (adx_header->block_size - 2) * 8 / adx_header->sample_bitdepth;
  int16_t scale[ adx_header->channel_count ];

  if (looping_enabled && !adx_header->loop_enabled)
     looping_enabled = false;

  // Повторяется до тех пор, пока не будет декодировано достаточное количество образцов, или не будет достигнут конец файла
  while (samples_needed > 0 && sample_index < adx_header->total_samples)
  {
     // Рассчитывает количество образцов, которое осталось декодировать в текущем блоке
     unsigned sample_offset = sample_index % samples_per_block;
     unsigned samples_can_get = samples_per_block - sample_offset;

     // Сокращает количество образцов, которые мы можем получить в этом проходе, если их количество не помещается в буфер (buffer)
     if (samples_can_get > samples_needed)
        samples_can_get = samples_needed;

     // Сокращает количество забираемых образцов, если поток недостаточно длинный, или рядом расположен переключатель цикла
     if (looping_enabled && sample_index + samples_can_get > adx_header->loop_end_index)
        samples_can_get = adx_header->loop_end_index - sample_index;
     else if (sample_index + samples_can_get > adx_header->total_samples)
        samples_can_get = adx_header->total_samples - sample_index;

     // Рассчитывает битовый адрес начала кадра, в котором расположен sample_index, и записывает это расположение
     unsigned long started_at = (adx_header->copyright_offset + 4 + \
                     sample_index / samples_per_block * adx_header->block_size * adx_header->channel_count) * 8;

     // Читает значения масштаба от начала каждого блока в этом кадре
     for (unsigned i = 0 ; i < adx_header->channel_count ; ++i)
     {
        bitstream_seek( started_at + adx_header->block_size * i * 8 );
        scale[i] = ntohs( bitstream_read( 16 ) );
     }

     // Предварительно рассчитывает значение останова для sample_offset
     unsigned sample_endoffset = sample_offset + samples_can_get;
```

```
      // Сохраняет адрес первого кадра в потоке бит сразу после масштаба в первом блоке кадра
      started_at += 16;
      while ( sample_offset < sample_endoffset )
      {
         for (unsigned i = 0 ; i < adx_header->channel_count ; ++i)
         {
            // Предсказывает следующий образец
            double sample_prediction = coefficient[0] * past_samples[i*2 + 0] + coefficient[1] * past_samples[i*2 + 1];

            // Переходит до смещения образца, читает и переписывает его расширяя до целочисленной переменной 32-х бит
            // Реализация знакового расширения остаётся в качестве упражнения для читателя
            // Знаковое расширение также необходимо для включения конечной подгонки, если в значении более 8 бит
            bitstream_seek( started_at + adx_header->sample_bitdepth * sample_offset + \
                            adx_header->block_size * 8 * i );
            int_fast32_t sample_error = bitstream_read( adx_header->sample_bitdepth );
            sample_error = sign_extend( sample_error, adx_header->sample_bitdepth );

            // Масштабирование значения коррекции ошибок
            sample_error *= scale[i];

            // Расчёт образца путём совмещения предсказания и коррекции ошибок
            int_fast32_t sample = sample_error + (int_fast32_t)sample_prediction;

            // Обновление последних образцов более новым образцом
            past_samples[i*2 + 1] = past_samples[i*2 + 0];
            past_samples[i*2 + 0] = sample;

            // Сокращение декодированного образца до верного диапазона для целочисленной переменной 16 -бит
            if (sample > 32767)
               sample = 32767;
            else if (sample < -32768)
               sample = -32768;

            // Сохранение образца в буфер (buffer) и увеличение переход на одну позицию
            *buffer++ = sample;
         }
         ++sample_offset;  // Мы декодировали по образцу из каждого блока, переводим указатель смещения блоков на 1
         ++sample_index;   // Это также означает, что у нас есть ещё один образец в потоке
         --samples_needed; // Также это означает, что осталось на одну меньше последовательностей образцов, которые нужно
декодировать
      }

      // Проверяем, не 'захватили' ли мы указатель окончания цикла, если да - переходим на начало цикла
      if (looping_enabled && sample_index == adx_header->loop_end_index)
         sample_index = adx_header->loop_start_index;
   }

   return samples_needed;
  }
```

Большей части этого кода должно быть достаточно для любого, кто разбирается в языке программирования Си.

Источник — https://ru.wikibooks.org/w/index.php?title=Язык_Си_в_примерах/Декодирование_звукозаписи_в_формате_ADX&oldid=139933

Эта страница в последний раз была отредактирована 26 октября 2017 в 13:28.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike, в отдельных случаях могут действовать дополнительные условия. Подробнее см. Условия использования.